|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Дарко Селаковић

**Поређење платформи отвореног кода за serverless рачунарство**

ДИПЛОМСКИ РАД

- Основне академске студије -

Нови Сад, 2023

|  |  |  |
| --- | --- | --- |
|  | Универзитет у Новом Сад, **Факултет техничких наука**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА** | Лист/Листова: |
|  |

*(Податке уноси предметни наставник - ментор)*

| Врста студија: | Основне академске студије |
| --- | --- |
| Студијски програм: | **Рачунарство и аутоматика** |
| Руководилац студијског програма: | **проф. др Милан Видаковић** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | **Дарко Селаковић** | Број индекса: | **RA 157/2019** |
| Област: | **Електротехника и рачунарство** | | |
| Ментор: | **проф. др Горан Сладић** | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; * литература | | | |

**НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:**

|  |
| --- |
| **Поређење платформи отвореног кода за serverless рачунарство** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| Анализирати принципе израде *serveless* апликација. Упознати се са концептима и платформама отвореног кода за *serverless* рачунарство. Специфицирати и имплементирати апликацију за пријаву гужве у саобраћају која ће служити за анализу. Документовати решење и запажања. |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Примерак за:  - Студента;  - Ментора |

САДРЖАЈ

[1. УВОД 7](#_Toc145328313)

[2. Примена *serverless* инфраструктуре 9](#_Toc145328314)

[2.1. Предности *serverless* инфраструктуре 10](#_Toc145328315)

[2.2. Мане *serverless* инфраструктуре 11](#_Toc145328316)

[2.3. Коришћени алати 13](#_Toc145328317)

[2.3.1. *Docker* 13](#_Toc145328318)

[2.3.1.1. *Docker* контејнер 14](#_Toc145328319)

[2.3.2. *Kubernetes* 15](#_Toc145328320)

[2.3.2.1. *Pod* 15](#_Toc145328321)

[2.3.2.2. *Minikube* 16](#_Toc145328322)

[3. Платформе отвореног кода 19](#_Toc145328323)

[3.1. Предности платформиотвореног кода 19](#_Toc145328324)

[3.2. Мане платформи отвореног кода 20](#_Toc145328325)

[3.3. Коришћене платформе 21](#_Toc145328326)

[3.3.1. *OpenFaaS* 21](#_Toc145328327)

[3.3.2. *Apache OpenWhisk* 24](#_Toc145328328)

[3.3.3. *Fission* 27](#_Toc145328329)

[4. Апликација за пријаву гужве у саобраћају 33](#_Toc145328330)

[4.1. Опис апликације 33](#_Toc145328331)

[4.2. Архитектура система 34](#_Toc145328332)

[4.3. Технологије и алати 36](#_Toc145328333)

[5. Имплементација система 37](#_Toc145328334)

[5.1. Нерегистровани корисник 37](#_Toc145328335)

[5.2. Регистровани корисник 44](#_Toc145328336)

[6. Анализа *serverless* платформи отвореног кода 53](#_Toc145328337)

[6.1. Анализа имплементације 53](#_Toc145328338)

[6.1.1. *OpenFaaS* 53](#_Toc145328339)

[6.1.2. *Apache OpenWhisk* 54](#_Toc145328340)

[6.1.3. *Fission* 55](#_Toc145328341)

[6.2. Анализа скалабилности и перформанси 56](#_Toc145328342)

[6.3. Сумирани резултати анализа 58](#_Toc145328343)

[7. ЗАКЉУЧАК 61](#_Toc145328344)

[ЛИТЕРАТУРА 63](#_Toc145328345)

[БИОГРАФИЈА 67](#_Toc145328346)

[КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА 69](#_Toc145328347)

[KEY WORDS DOCUMENTATION 71](#_Toc145328348)

# 1. УВОД

Последњих година платформе отвореног кода (енгл. *open source*) све више добијају на популарности. То је последица лакшег прилагођавања потребама самог корисника и због своје транспарентности, која није у толикој мери присутна код комерцијалних производа. Ова транспарентност највише долази до изражаја јер је корисницима омогућен слободан приступ изворном коду, који осим што могу да посматрају, такође могу и да модификују и унапређују. Велика предност ових платформи је и у томе што најчешће пружају велику заједницу корисника, која омогућава лакше решавање проблема. Такође будући да су ове платформе бесплатне корисници не морају плаћати лиценце и претплате.

Инфраструктура за рачунарство без сервера (енгл. *Serverless computing*) омогућава програмерима да сву своју пажњу усмере на писање кода који ће затим да имплементирају у контејнере којима управља давалац услуга (енгл. *provider*) рачунарства у облаку. Давалац услуга се даље брине о свему потребном, као што је инфраструктура неопходна за покретање кода и њено одржавање, ажурирање, управљање сигурношћу и многе друге функције [1].

Мотивација за настанак платформи отвореног кода за *serverless* рачунарство произилази из потребе за лако доступним, поузданим алатима који ће омогућити брз развој *serverless* апликација. Ови софтвери пружају велику заједницу програмера који својим радом стварају иновативна решења и на тај начин проширују своја знања и стварају платформе које пружају решења за различите типове корисника.

Овај рад је фокусиран на поређење платформи отвореног кода за *serverless* рачунарство.

У наставку рада биће анализирано неколико оваквих платформи, истражене карактеристике, функционалности и перформансе. Анализа је урађена на примеру апликације за пријаву гужве у саобраћају која се састоји од клијентске апликације и три *serverless* платформе (*OpenFaaS, Fission, Apache OpenWhisk*) које имплементирају пословну логику за апликацију и обрађују захтеве које им клијентска апликација шаље. Такође су истражене и међусобно упоређене карактеристике, перформансе и могућности сваке од ових платформи.

У другом поглављу биће речи о применама *serverless* инфраструктуре, њеним предностима и манама, као и о алатима који су коришћени за подешавање окружења и имплементацију платформи. У трећем поглављу представљене су предности платформи отвореног кода и њихови недостаци и ближе су објашњене платформе коришћене приликом израде овог дипломског рада. У четвртом поглављу описана је апликација, архитектура система као и технологије и алати који су употребљени. У петом поглављу приказана је и размотрена имплементација система на све три платформе. Имплементација апликације на платформама, као и скалабилност и перформансе сваке од платформи анализиране су у петом поглављу и сумирани су резултати анализе. У последњем поглављу сумирани су закључци до којих се дошло током израде апликације.

# 2. Примена *serverless* инфраструктуре

*Serverless computing* омогућава програмерима да усмере сву своју пажњу на писање кода за кориснички интерфејс и пословну логику. Све што програмери треба да ураде јесте да напишу свој код апликације и имплементирају га у контејнере којима управља давалац услуга рачунарства у облаку. Давалац услуга рачунарства у облаку се брине за све остало, обезбеђујући инфраструктуру потребну за покретање кода и динамички скалирајући инфраструктуру према захтевимa. Давалац услуга рачунарства у облаку такође је одговоран за рутинско управљање и одржавање инфраструктуре, као што су ажурирања и закрпе оперативног система, управљање сигурношћу, планирање капацитета, праћење система и много више [1].

*Serverless* архитектура је идеална за асинхроне апликације без стања које могу бити одмах покренуте. Такође, *serverless* архитектура је добар избор за случајеве коришћења који доживљавају ретке и непредвидиве скокове у захтевима. Добар пример таквог случаја коришћења је апликација за периодичну обраду велике количине долазних сликовних датотека за које у тренутку када стигну апликација мора бити спремна. *Serverless* апликације се такође користе у случајевима који укључују долазне токове података, ботове за ћаскање, заказане задатке или пословну логику. Неке друге честе употребе *serverless* технологије су серверска страна *Application Programming Interface* (API) *web* апликације, аутоматизација пословних процеса, *serverless* *web* сајтови и интеграција између више система [2].

Рачунарство без сервера се разликује од традиционалних концепата рачунарства у облаку у смислу да су инфраструктура и платформе на којима услуге функционишу скривене од корисника. У овом приступу, корисници су заинтересовани само за жељену функционалност њихове апликације, док се остали аспекти делегирају пружаоцу услуге. Велике технолошке компаније као што су Амазон, Гугл (енгл. *Google*) и Мајкрософт (енгл. *Microsoft*) нуде *serverless* платформе под различитим именима. Иако се детаљи услуга могу разликовати, основна идеја иза понуђених услуга је готово иста, а то је пружање услуга према моделу плаћања по потрошњи. *Serverless* инфраструктура настоји постићи аутоматско скалирање уз пружање повољних услуга [32].

*Function as a Service* (FaaS) је модел *serverless* рачунарства у којем програмери пишу и извршавају функције као појединачне јединице кода, уместо традиционалних монолитних апликација. Програмери могу писати функције у стандардним програмским језицима, што олакшава развој и омогућава широк спектар језика за избор. Функције се могу имплементирати у облаку, што значи да се извршавају на инфраструктури која је обезбеђена од стране пружаоца услуге облака. Програмери се не морају бринути о управљању инфраструктуром или одржавању сервера. Ово смањује административни терет и омогућава фокусирање на писање кода. FaaS је популаран у различитим апликацијама, укључујући развој апликација и обраду података у стварном времену где се брзо скалирање и ефикасна употреба ресурса сматрају кључним предностима [81].

## 2.1. Предности *serverless* инфраструктуре

*Serverless* инфраструктура доноси бројне предности и могућности за ефикасно извршавање и скалирање апликација. Неке од тих предности су:

* **Унапређено управљање ресурсима –** у класичном моделу рачунарства у облаку, корисник сам бира и имплементира одговарајуће ресурсе који су обично ограничени на велике, вишефункционалне врсте ресурса, углавном контејнере. Међутим, ови ресурси обично нису оптимално прилагођени апликацијама, што може довести до смањене ефикасности. Како би се то избегло, апликације су често грубо подељене. Насупрот томе, концепт *serverless* рачунарства подразумева разбијање апликација на много мање компоненте и на тај начин пружа прецизније усклађену, апстрактну потражњу ресурса за стварним физичким ресурсима система [12].
* **Већи увид и контрола –** *serverless* рачунарство пружа нову перспективу на начин управљања апликацијама у облаку. Уместо да корисници сами изаберу и скалирају ресурсе, *serverless* модел премешта одговорност на оператeре. Они имају већу контролу над одабиром, постављањем и праћењем ресурса. Осим тога, *serverless* рачунарство омогућава анализу и моделирање финих делова апликација, што доприноси бољим одлукама у вези са скалирањем, праћењем оптерећења и прилагођавањем ресурса. Овакав приступ смањује оптерећење корисника и омогућава оператерима да пруже ефикаснију и прецизнију услугу, омогућавајући тако боље искоришћење ресурса и побољшање перформанси апликација [12].
* **Ситно скалирање –** предност *serverless* инфраструктуре је у могућности финог скалирања. Док традиционални модели често захтевају скалирање целих апликација или виртуелних машина, *serverless* омогућава скалирање појединачних услуга или функција. То значи да се ресурси могу тачно прилагодити потребама одређених делова апликација, без потребе за скалирањем целокупног система. Ово омогућава ефикасније искоришћење ресурса и боље перформансе, а такође и мање захтева за стручношћу корисника [12].
* **Програмски језици** – на оваквим платформама, могуће је користити различите програмске језике као што су *JavaScript* [39], *Java* [40], *Python* [26], *Go* [41], *C#* [42] и *Swift* [43] како би се имплементирале различите функционалности. Већина ових платформи подржава више од једног језика, а неке чак омогућавају и проширивање функционалности кода написаног на било ком језику [13].
* **Извршавање кода** – ове платформе су створене како би омогућиле извршавање кода на што једноставнији начин. Уместо да програмери морају доставити велику количину података, често је довољно да обезбеде само датотеку са изворним кодом функције коју желе покренути. Такође, постоји неколико опција како тај код може бити организован и спакован. На пример, програмери могу креирати архиву са више датотека или користити *Docker* слику са већ припремљеним кодом. Уз то, ту су и могућности за груписање више функција заједно и праћење различитих верзија истих функција. Иако се ове опције не користе често, могу бити веома корисне за бољу организацију и управљање пројектима [13].
* **Остали бенефити –** осим већ поменутих предности, *serverless* инфраструктура доноси и широк спектар других предности које додатно подржавају њено прихватање. На пример, прелазак са фиксних инвестиција на променљиве трошкове омогућава боље усклађивање финансија са реалним потребама пословања. Такође, способност коришћења независних услуга омогућава тимовима да бирају оптималне алате за специфичне ситуације, без нарушавања функционалности других делова система или саме организације. Захваљујући високом нивоу апстракције, програмери могу брже развијати и тестирати апликације у оваквим дистрибуираним окружењима, смањујући потребу за дубоким техничким познавањем дистрибуираних система. Ова прилагодљивост убрзава развојни процес и доприноси бољој искоришћености ресурса, омогућавајући брже испоруке решења [12].

## 2.2. Мане *serverless* инфраструктуре

Иако *serverless* инфраструктура пружа бројне предности, такође има нека ограничења и изазове. У овом делу биће размотрене неке од главних мана ове инфраструктуре:

* **Ограничење за неке апликације** – за кoриснике FaaS модел који платформе пружају може имати своја ограничења која не одговарају свим врстама апликација. На пример, могуће је да одређена платформа не подржава најновију верзију *Python* програмског језика или да неке посебне библиотеке које су потребне нису доступне на њој [13].
* **Ограничења за добављаче услуга** – данашње потребе захтевају решавање различитих изазова као што су управљање животним циклусом корисничких функција, постизање способности за скалирање и осигурање отпорности система на грешке. Ово такође подразумева да програмери морају пажљиво разумети како се платформа понаша и прилагодити дизајн апликације тако да одговара овим специфичним карактеристикама платформе [13].
* **Асинхрони позиви** – коришћење асинхроних позива ка *severless* функцијама, као и комуникација између њих, може додатно да закомпликује структуру система. Уобичајено је да удаљени API позиви прате образац „захтев-одговор“, што се лакше постиже са синхроним позивима *severless* функција. Како би се ово поједноставило користе се синхрони позиви кад год је то могуће. Додатно, како би се корисници обавестили о успеху операције током коришћења асинхроних позива често се користи интеграција путем редова порука. Међутим, важно је напоменути да асинхрони позиви нису увек најпогодније решење за сваку ситуацију. Они се посебно добро примењују за једнократне задатке, као што је на пример покретање дуготрајног процеса израде резервне копије [14].
* **Позиви функција другим функцијама** – ова радња носи са собом изазове у детектовању сложених грешака и има ограничену способност изолације функционалности. Ово може додатно повећати трошкове ако се функције позивају синхроно, јер захтева плаћање за истовремено извршавање обе функције. Ради избегавања овога, препоручује се избегавање позивања једне функције унутар друге или спајање функција кад год је то могуће [14].
* **Дељени код између функција** – могуће је да се постојеће *severless* функције које зависе од дељеног кода могу пореметити услед измена тог заједничког кода. Такође, постоји опасност да се премаши максимална дозвољена величина слике (обично 500MB на већини платформи) и да се продужи време потребно за покретање функције (што је слика већа, то ће дуже трајати покретање саме функције). Да бисмо избегли ове ризике, препоручује се приступ стварања функција које су независне и изоловане једна од друге. Такође, добро је применити чисту архитектуру и користити заједнички код путем јасно дефинисаних и тестираних интерфејса [14].
* **Употреба превише библиотека** – употреба већег простора услед коришћења различитих библиотека повећава ризик од прекорачења дозвољене величине слике и успорава време покретања функције. Како би се то избегло, препоручује се да се увозе само оне библиотеке које су заиста неопходне, како би се избегло оптерећење система. У *AWS Lambda* окружењу [45], постоји могућност да се библиотеке учитавају приликом покретања из спољног складишта и смештају у привремени директоријум, што може бити корисно, нарочито ако брзина покретања није од пресудног значаја [14].
* **Превише функција** – проблем се јавља када се уместо већ постојећих функција стварају нове *serverless* функције, што није увек неопходно. Битно је напоменути да неактивне *serverless* функције не генеришу трошкове, што може иницирати креирање нових функција уместо прилагођавања постојећих. Међутим, ово може довести до компликација у одржавању система. Решење за овај изазов је пажљиво разматрање потребе за новом функцијом и груписање функција у микросервисе како би се олакшала интеракција између услуга [14].
* **Трансформација старог кода** – један од недостатака *severless* инфраструктуре, јесте изазов преласка са постојећег кода на нове *severless* функције. Трансформација старијег кода у нови код који је компатибилан са *serverless* окружењем захтева значајно улагање ресурса и труда. Овај процес може бити компликован и захтевати доста времена, с обзиром на потребу за мењањем већ постојећег кода [13].

## 2.3. Коришћени алати

За подешавање окружења и имплементацију платформи отвореног кода за *serverless* рачунарство важни су алати који се за то користе. Ови алати играју кључну улогу у омогућавању ефикасног развоја, тестирању и скалирању *serverless* апликација. Поред одговарајућих алата, важно је разумети и друге аспекте који утичу на успешност ових технологија. Неки од тих аспеката су подршка за различите програмске језике, интеграција са другим сервисима и алатима, као и могућност аутоматског скалирања и управљања ресурсима. Различити алати могу подржавати различите програмске језике, па је важно одабрати оне који подржавају језике које користимо у развоју апликација. Поред ових фактора, треба водити рачуна и о документацији, безбедности и доступности заједнице за подршку.

### 2.3.1. *Docker*

*Docker* [3] је платформа отвореног кода која покреће апликације и олакшава њихов развој и дистрибуцију. Апликације које су изграђене на *Docker* платформи су запаковане са свим потребним зависностима у стандардни облик назван контејнер. Иако технологије контејнера постоје већ више од 10 година, *Docker*, као релативно нова иновација, тренутно је једна од најуспешнијих технологија, јер доноси нове могућности које претходне технологије нису имале. Програмер може једноставно запаковати апликације у лагане *Docker* контејнере. Ове виртуелизоване апликације се могу лако изводити било где без икаквих измена. Осим тога, *Docker* може подржати више виртуелних окружења него друге технологије на истом хардверу. Као крајњи резултат, *Docker* се лако може интегрисати са алатима треће стране, што олакшава имплементацију и управљање *Docker* контејнерима. *Docker* контејнери се лако могу имплементирати у окружењима рачунарства у облаку [33].

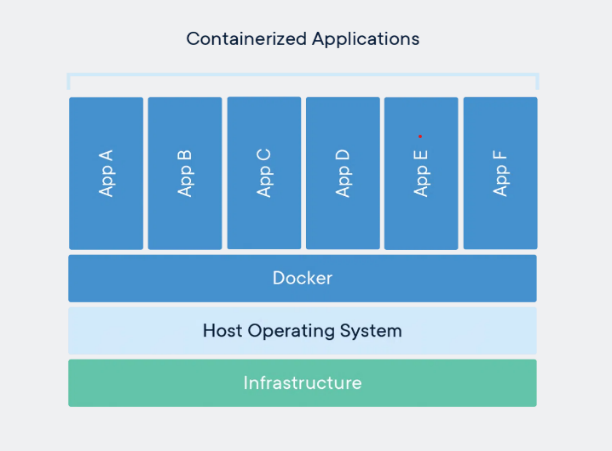
*Docker* уклања понављајуће и замарајуће задатке конфигурације и користи се током целог животног циклуса развоја за брз, једноставан и преносив развој апликација за *desktop* и за рачунарство у облаку. Свеобухватна *Docker* платформа од почетка до краја укључује графичке корисничке интерфејсе, командне линије и API који су дизајнирани да заједно функционишу током целог процеса развоја апликација [3]. Развој апликација данас захтева много више од писања кода. Више језика, радних оквира, архитектура и неповезаних интерфејса између алата за сваку фазу животног циклуса ствара огромну сложеност. *Docker* поједностављује и убрзава радни ток, истовремено пружајући програмерима слободу да иновирају користећи своје изабране алате, слојеве апликација и окружења за имплементацију за сваки пројекат [4].

#### 2.3.1.1. *Docker* контејнер

*Docker web* страница дефинише Docker контејнер као „стандардну јединицу софтвера која пакује код и све његове зависности како би апликација брзо и поуздано радила у различитим рачунарским окружењима. *Docker* контејнер слика је лагани, самостални, извршни пакет софтвера који укључује све што је потребно да се покрене апликација: код, извршно окружење, системске алате, системске библиотеке и подешавања. Контејнерске слике постају контејнери када се извршавају на *Docker* платформи“ [5]. Доступан за *Linux* [46] и *Windows* [47] апликације, контејнеризовани софтвер ће увек радити исто, без обзира на инфраструктуру. Контејнери изолују софтвер од окружења и осигуравају да он ради једнако без обзира на разлике, на пример између фазе израде и фазе продукције [5].

*Docker* слика креира *Docker* контејнер. Контејнери садрже све што је потребно за једну апликацију, тако да се апликација може покретати изоловано. На пример, претпоставимо да постоји слика *Ubuntu* [48] оперативног система са *Structured Query Language* (SQL) [49] сервером. Када се ова слика покрене командом *docker run*, тада ће се креирати контејнер и SQL сервер ће се покретати на *Ubuntu* оперативном систему [33].

Слика 2.1. приказује инфраструктуру апликације у чијем развоју се користe Docker контејнери.



Слика 2.1. *Docker* контејнер инфраструктура [[1]](#footnote-1)

### 2.3.2. *Kubernetes*

*Kubernetes* [6], такође познат као K8s, је платформа отвореног кода за аутоматизацију имплементације, скалирање и управљање контејнеризованим апликацијама. Он групише контејнере који чине апликацију у логичке јединице ради лакшег управљања и откривања. *Kubernetes* постепено уводи промене у апликацију или њену конфигурацију, пратећи здравље апликације како би осигурао да се не уклонe све инстанце истовремено. Уколико нешто крене наопако, *Kubernetes* ће поништити промену. Поново покреће контејнере који не успевају, замењује и поново распоређује контејнере када чворови престану да раде, зауставља контејнере који не одговарају на кориснички дефинисане провере здравља и не оглашава их клијентима док не буду спремни да пруже услугу [6].

Док је *Docker* покретач контејнера, *Kubernetes* је платформа за покретање и управљање контејнерима из различитих покретача контејнера. *Kubernetes* подржава бројне покретаче контејнера, укључујући *Docker*, *containerd* [50] и било коју имплементацију *Kubernetes Container Runtime Interface* (CRI) [51]. Добра метафора је да је *Kubernetes* оперативни систем, а *Docker* контејнери су апликације које се инсталирају на оперативни систем [7].

Ресурси у *Kubernetes*-у могу бити креирани директно путем командне линије, али се обично дефинишу користећи *Yet Another Markup Language* (YAML) [34, 52].

#### 2.3.2.1. *Pod*

*Pod* [52] представља групу једног или више контејнера. *Kubernetes* распоређује све контејнере унутар једног *pod*-а на истом чвору, са истим мрежним именским простором, тако да сви имају исту *Internet* *Protocol* (IP) адресу и могу међусобно приступити користећи *localhost*. Пример дефинисања *pod*-a се може видети на листингу 2.1. [34]

На листингу 2.1. дефинисани *pod* има два контејнера. Један контејнер користи *Nginx web* сервер и може читати податке са *„/srv/www“* локације, док други контејнер са називом *„git-monitor“* има приступ *Git* складишту и такође може приступити истом директоријуму кроз *„emptyDir“* диск*.*

Листинг 2.1. Пример дефинисања *pod*-a

apiVersion: v1

kind: Pod

spec:

containers:

- name: nginx

image: nginx

volumeMounts:

- mountPath: /srv/www

name: www-data

readOnly: true

- name: git-monitor

image: kubernetes/git-monitor

- name: GIT\_REPO

value: http://github.com/some/repo.git

volumeMounts:

- mountPath: /data

name: www-data

volumes:

- name: www-data emptyDir: {}

Не само да контејнери унутар *pod*-а могу делити исте мрежне интерфејсе, већ могу делити и исте податке, као што је приказано на листингу 2.1.

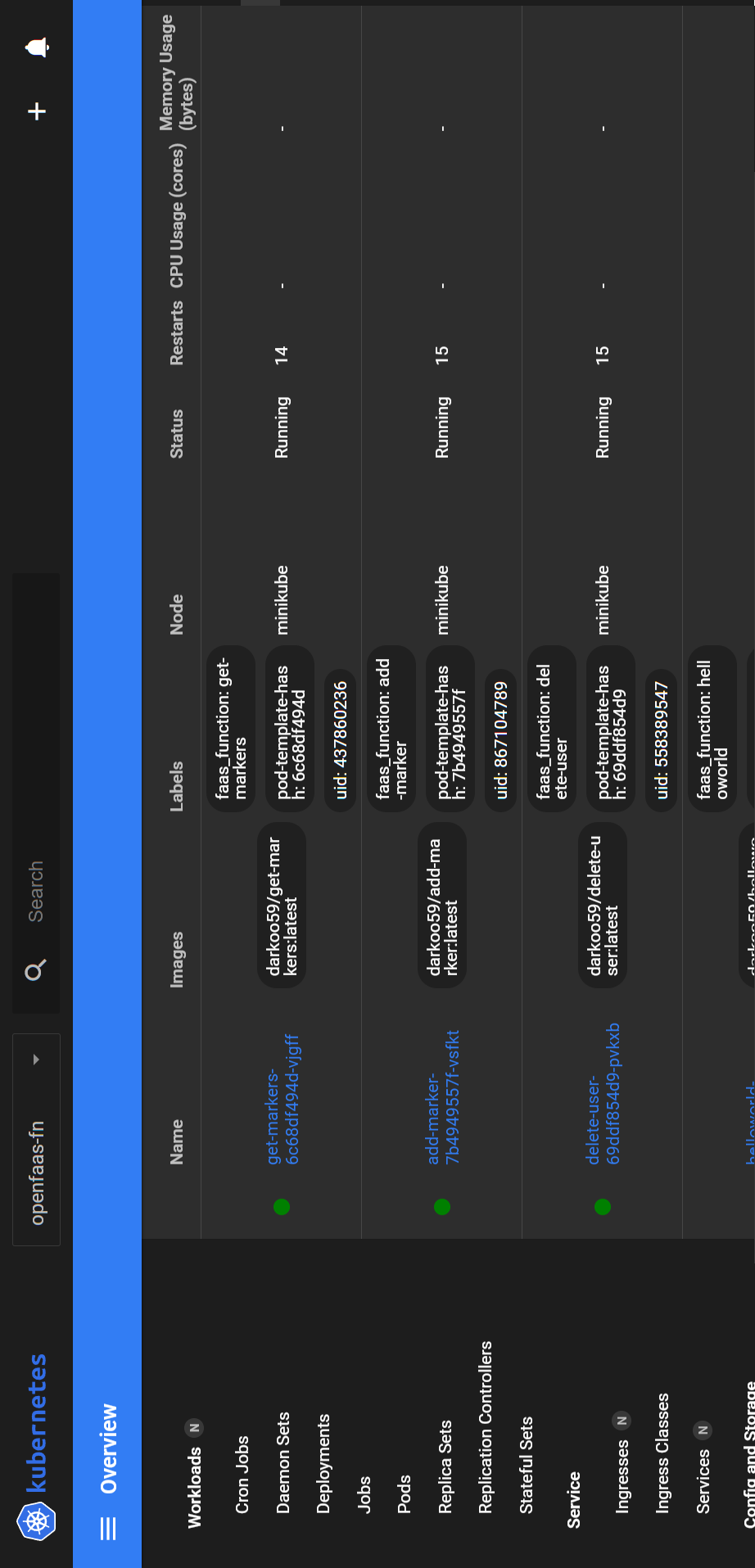
Када се *pod* креира, *Kubernetes* ће га надгледати и аутoматски га поново покренути ако се процес прекине. Поред тога, *Kubernetes* се може конфигурисати да покуша да се повеже са контејнером преко мреже како би утврдио да ли је *pod* спреман (*readinessProbe*) и да ли је још увек активан (*livenessProbe*) [34].

#### 2.3.2.2. *Minikube*

*Minikube* [8] брзо поставља локални *Kubernetes* кластер на *macOS* [54], *Linux* и *Windows* оперативне системе. Фокусира се на помоћ програмерима и новим корисницима *Kubernetes* система [8].

*Minikube* користи виртуелизациони софтвер као што су *VirtualBox* [55], *VMware* [56] или *KVM* [57] како би покренуо кластер. Након што је *Minikube* инсталиран, може се користити командна линија да се изврше различите команде [34]:

* ***minikube start*** – покретање кластера
* ***minikube stop*** – заустављање кластера
* ***minikube ip*** – утврђивање IP адресе кластера
* ***minikube logs*** – преглед logova
* ***minikube dashboard*** – отварање контролне табле кластера у *web* претраживачу, као што се може видети на слици 2.2.



Слика 2.2. *Minikube* контролна табла

# 

# 3. **Платформе отвореног кода**

Постоји велики број платформи отвореног кода за *serverless* рачунарство, које пружају различите могућности и функционалности за развој и управљање *serverless* апликацијама. Свака од ових платформи има своје предности и мане, а која ће се користити највише зависи од захтева самог клијента и специфичности пројекта на коме се ради.

У наставку биће описане платформе које су коришћене у овом раду. Свака од ових платформи ће бити анализирана по различитим критеријумима, као што су доступност, скалабилност, флексибилност, подржане технологије, документација, подршка заједнице и остали важни критеријуми.

## 3.1. Предности платформиотвореног кода

Изван контекста истраживања, постоји много убедљивих разлога зашто аутори користе платформу отвореног кода. Међу њима се истичу очекивања да ће понуда бесплатног кода повећати тржишни удео и веровање да ће изградња заједнице развоја око алата повећати дугорочну одрживост, као и да ће повећати лични углед аутора. За оне који не желе да комерцијализују своје софтверске производе, отворен код нуди низ потенцијалних предности [15]:

* **Услови лиценце** – изворни код је отворен и доступан свим корисницима. Услови лиценце су обично неутрални и непристрасни према даваоцу или примаоцу лиценце. Једноставни и прецизни услови олакшавају усклађивање и не захтевају праћење употребе лиценце [16].
* **Флексибилност** - изворни код програма доступан је свима и то омогућава његово прилагођавање и измену. Ова доступност изворног кода олакшава другим људима да изврше промене и адаптације. Употреба оваквих софтверских решења омогућава више могућности за доношење одлука, не само у оквиру софтвера, већ и у контексту подршке и хардвера. Већи број опција омогућава већу флексибилност, што представља предности овог модела [16].
* **Смањење грешака** – честе мале измене и обимно тестирање кода смањују грешке. У случају отвореног кода, широко тестирање има велики успех јер много корисника може детектовати проблеме. Што је више људи којим је код доступан, веће су шансе да неко уочи грешку. Постоји и слобода у избору начина решавања проблема, што мотивише тимове на њихово решавање [17].
* **Мотивација** – заједница је за програмере веома битна. Они стичу самопоуздање на основу постигнутих резултата. С гледишта овог аспекта, извор мотивације лежи у учешћу у пројектима и савладавању изазова који се појављују у току пројеката [17].
* **Одсуство притиска код развојног тима** – у планирању оваквих пројеката примећује се неформалан приступ. Уместо конкретних планова и детаљних визија, фокус је на дугорочном напретку пројекта. Установљено је да многи такви пројекти функционишу веома ефикасно, посебно када је рад заснован на добровољном ангажовању. Учесници нису обавезани да раде под пуним радним временом, што одстрањује притисак који налажу традиционални планови [17].

## 3.2. Мане платформи отвореног кода

Поред присутних бројних предности, као и код било које технологије, платформе отвореног кода нису без мана. У овом делу рада биће истражене неке од најчешћих мана, као и бројни недостаци, како би стекли целовиту слику потенцијалних слабости ових платформи:

* **Контрола квалитета** – пројекти који користе платформе отвореног кода обично се ослањају на критике и одзив заједнице за контролу квалитета. Критике заједнице понекад не успевају открити суштинске грешке у архитектури. Такође, нису довољно ефикасне у откривању скривених грешака. Овакве грешке ретко настају и тешко их је приметити током процеса тестирања кода. Један појединац може идентификовати грешку, док остали могу потрошити дуже време у тражењу исте, на основу недостатка информација. Због свега овога, важно је користити разноврсне методе тестирања и контроле квалитета како би се осигурала стабилност и висок квалитет пројекта [17].
* **Документација** – пројекти који се користе отвореним кодом обично улажу значајне напоре у разраду детаљне документације. Интересантно је да често може бити изазовно омогућити довољно ресурса за одржавање исте. Као резултат, многе информације о знању и вештинама на пројекту остају незабележене, што може знатно усложнити процес учења о тим пројектима и притом изискује више времена [17].
* **Сигурност** – с обзиром на отвореност кода, сви корисници имају приступ изворном коду, што омогућава брже откривање и решавање проблема. Ипак, ова отвореност такође значи да потенцијално и злонамерни корисници могу прегледати код и пронаћи раније невидљиве слабости у циљу злоупотребе. У сваком случају, корисници имају могућност да проуче и оцене стварни ниво безбедности пре него што усвоје производ [16].

## 3.3. Коришћене платформе

### 3.3.1. *OpenFaaS*

*OpenFaaS* [9] омогућава програмерима једноставно имплементирање функција и микросервиса на *Kubernetes* платформи, без понављајућег и досадног кодирања. Паковањем кода или већ постојећег бинарног фајла у *Docker* слику добија се високо скалабилна крајња тачка са аутоматским скалирањем и метрикама [9].

Главне карактеристике су следеће [9]:

* Платформа за преносиве функције – извршавање функције на било којој локалној или инфраструктури рачунарства у облаку
* Могућност писања функција на било ком језику и запакивања у *Docker*/*Oracle Cloud Infrastructure* (OCI) форматиране контејнере
* Једноставност за коришћење – уграђени кориснички интерфејс, моћна командна линија и инсталација једним кликом
* Прилагођено скалирање према потреби – могућност ношења са наглим порастом саобраћаја и скалирање у неактивном стању
* *OpenFaaS Pro* за комерцијалну употребу и производњу
* *Community Edition* за истраживање функционалности или изградњу решења за доказ концепта (енгл. *Proof of Concept*) без икаквих трошкова

Заснован на *Kuberntes* систему, *OpenFaaS* поједностављује управљање сложеним апликацијама са више функција. Да би постигао брзо време покретања и гашења, инстанце функција се изводе унутар контејнера. API *gateway*, једна од главних компоненти *OpenFaaS* платформе, пружа интерфејс за креирање, брисање, измену, праћење и скалирање функција. Такође је одговоран за прихватање екстерних и интерних захтева и усмеравању истих ка одговарајућој функцији ради обраде [35].

Платформа се ослања на *Kubernetes* за оркестрацију и управљање *serverless* функцијама. Конкретно, *Kubernetes* се користи за откривање услуга, аутоматско скалирање, балансирање оптерећења и рутирање мреже. Свака функција се такође имплементира помоћу *pod*-a. *OpenFaaS* омогућава развој сложених *serverless* апликација као композицију аутономних и одвојених функција. Ток рада апликације је дефинисан у оркестрационој функцији. На тај начин, укључене функције остају несвесне свог састава. На платформи, API *gateway* управља и скалира функције, излаже их путем *Representational State Transfer* (REST) [58] АPI-ја и прикупља метрике на нивоу функција путем *Prometheus* алата [59]. За разлику од других алтернатива, као што је *Apache OpenWhisk* [18], *OpenFaaS* омогућава скалирање функција до нуле [35].

Функције се могу писати на било ком језику и уграђене су у преносиве *Oracle Cloud Infrastructure* (OCI) слике. Постоје званични шаблони за: *Go*, *Java*, *Python*, *C#*, *Ruby* [60], *Node.js* [61], *PHP* [62], а може се написати и сопствени шаблон. Могу се користити и постојећи микросервиси написани у различитим радним оквирима као што су *Express.js* [63], *Vert.x* [64], *Flask* [27], *ASP.NET Core* [65], *FastAPI* [66] и *Django* [9, 67].

*OpenFaaS* платформа је написана у програмском језику *Go* и има велику заједницу корисника који користе платформе отвореног кода. То може значити пружање повратних информација кроз тестирање функционалности, предлагање побољшања или укључивање у одржавање скоро 50 пројеката. Језгро *OpenFaaS* платформе је независни пројекат који користи платформу отвореног кода коју је оригинално креирао Алекс Елис 2016. године. Сада се пројекат развија и обликује од стране све веће заједнице сарадника и корисника [10].

Постоје три препоручена начина за инсталацију *OpenFaaS*-a на *Kubernetes* кластер [10]:

* Коришћењем *Command Line Interface* (CLI) алата за инсталацију *arkade* – (препоручено)
* Помоћу *Helm* [69], *Flux* [70] или *ArgoCD* алата (*GitOps* радни ток) [71]
* Коришћењем статички генерисаних YAML фајлова - (није препоручено)

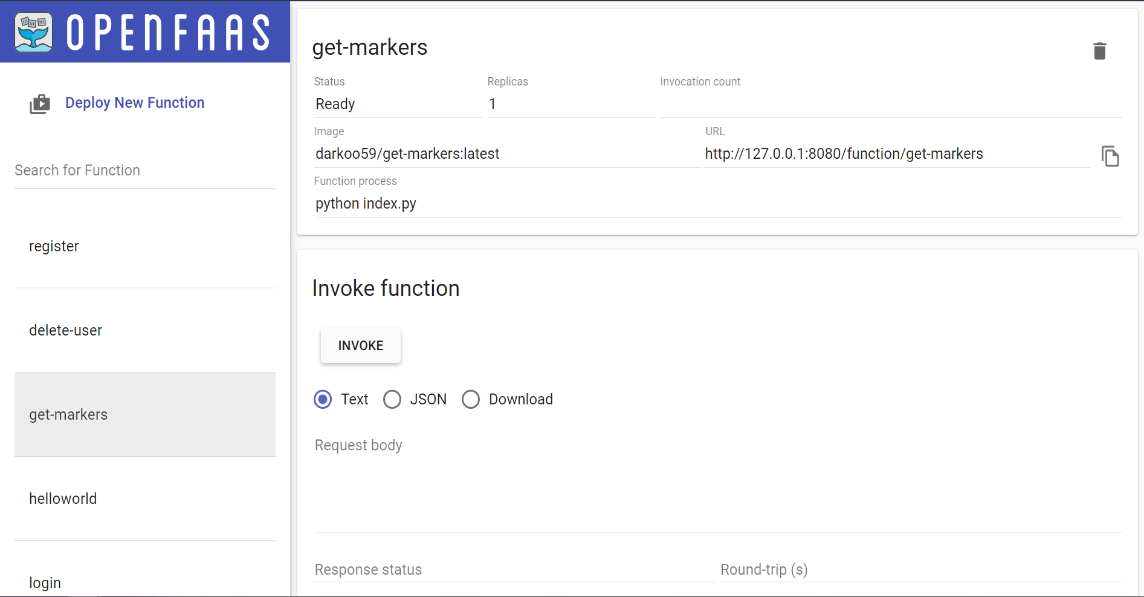
Званични CLI за *OpenFaaS* је *faas-cli*. CLI алат се може користити за изградњу и имплементацију функција на *OpenFaaS* платформи. Могуће је изградити *OpenFaaS* функције користећи подржане језичке шаблоне. То значи да само треба написати фајл за руковођење (као што је *handler.py/handler.js*), а CLI алат ће обавити остатак посла креирања *Docker* слике [11]. Препорука је да се *faas-cli* инсталира уз помоћ *arkade* алата, за најбржу и најлакшу инсталацију [10].

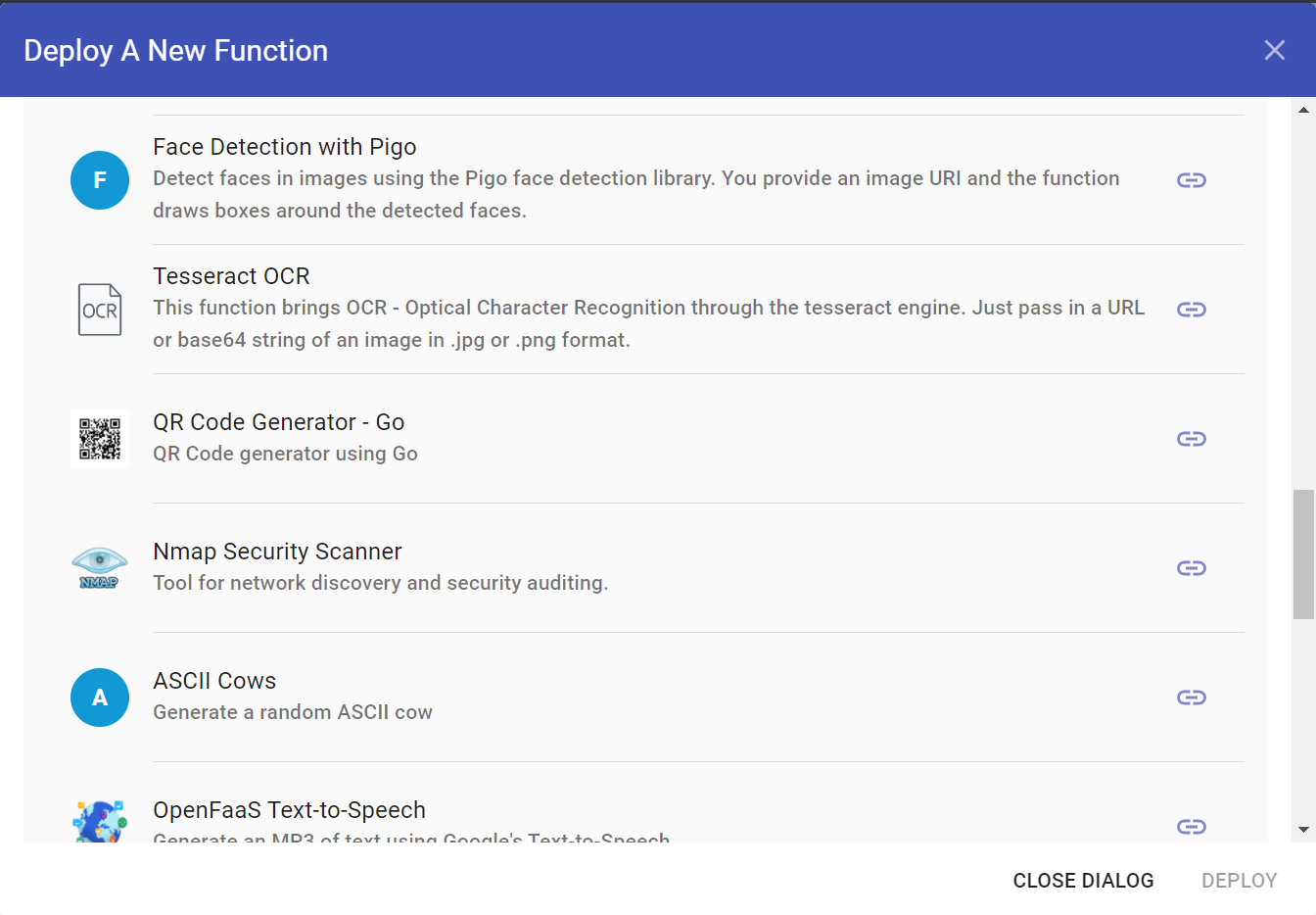
Након што се инсталира *faas-cli*, може се почети са креирањем и имплементирањем функције путем следећих наредби [10]:

* *faas-cli build* – изградња слике у локалној *Docker* библиотеци
* *faas-cli push* – слање те слике на удаљени регистар контејнера
* *faas-cli deploy* – имплементација функција на кластеру
* *faas-cli up* – аутоматизује све горенаведене кораке у једној наредби

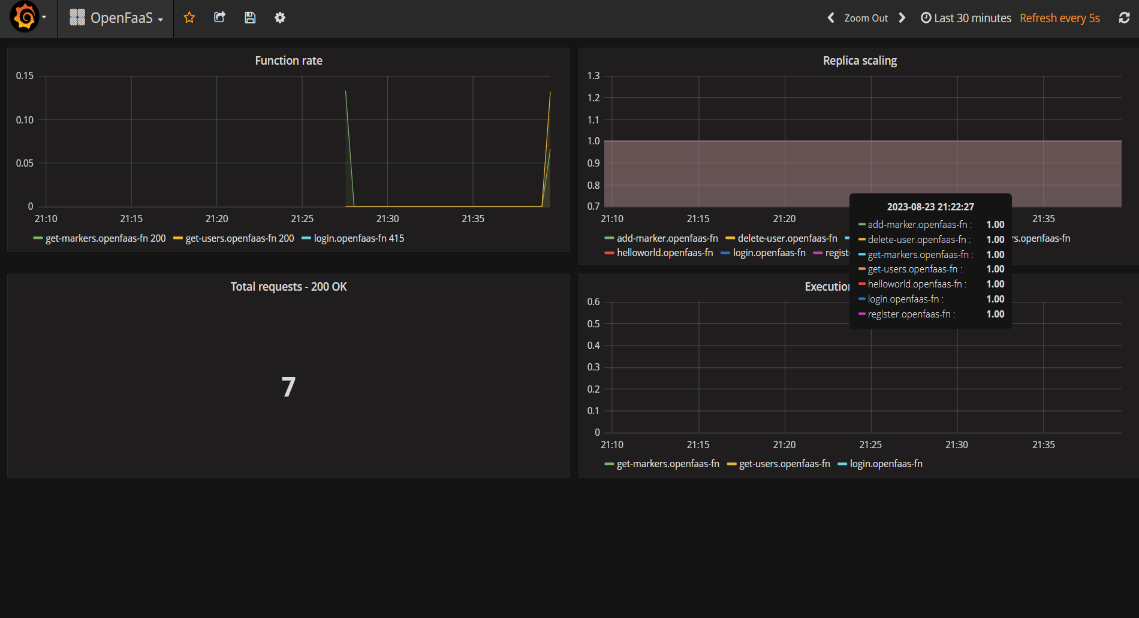
*OpenFaaS Dashboard* је нови кориснички интерфејс који је редизајниран како би управљање и разумевање *OpenFaaS* платформе било једноставније [10].

На слици 3.1. приказан је изглед *OpenFaaS Dashboard* корисничког интерфејса.

Слика 3.1. *OpenFaaS Dasboard*

 Велика предност платформе у поређењу са осталим платформама је и могућност креирања и коришћења шаблона за олакшану израду функција. Корисници могу генерисати шаблоне користећи терминал или избором шаблона са корисничког интерфејса, као што је то приказано на слици 3.2.

Слика 3.2. Приказ доступних шаблона на *OpenFaaS Dashboard* корисничком интерфејсу

 *OpenFaaS* такође аутоматски прати метрике функција користећи *Prometheus*. Метрике се могу претворити у корисну надзорну таблу уз помоћ бесплатних и алата који се користе на платформама отвореног кода као што је *Grafana* [72], као што је то приказано на слици 3.3.

Слика 3.3. Приказ праћења метрика путем алата *Grafana*

### 3.3.2. *Apache OpenWhisk*

*Apache OpenWhisk* је платформа отвореног кода, дистрибуирана *serverless* платформа која извршава функције као одговор на догађаје без обзира на њихов обим. *OpenWhisk* управља инфраструктуром, серверима и скалирањем користећи *Docker* контејнере, омогућавајући фокусирање на изградњу сигурних и ефикасних апликација [18].

*OpenWhisk* платформа подржава програмски модел у којем програмери пишу функционалну логику (познату као акције) у било којем подржаном програмском језику, која се динамички распоређује и покреће као одговор на повезане догађаје (путем окидача) из спољних извора или путем *Hypertext Transfer Protocol* (HTTP) захтева. Пројекат укључује CLI базиран на REST API-ју, заједно са другим алатима за подршку паковања, каталог услуга и многе популарне опције за имплементацију контејнера [18].

*Apache OpenWhisk* може бити имплементиран и конфигурисан на разним платформама, јер своје компоненте гради користећи контејнере, што му омогућава подршку за различите опције имплементације, како локално, тако и унутар инфраструктуре рачунарства у облаку. Имплементација се може извршити на платформама као што су *Kubernetes*, *Mesos* [72] и *OpenShift* [36, 73]. Општеприхваћено је да заједница подржава имплементацију на *Kubernetes* систему помоћу *Helm* шаблона, јер пружа много једноставних и практичних решења како за програмера, тако и за операторе [18].

Програмски модел платформе се заснива на три основна елемента, а то су акције, окидачи и правила. Акције представљају функције које извршавају произвољан код. Окидачи представљају класу догађаја који долазе из различитих извора. Правила омогућавају повезивање окидача са акцијама. Поред ових основних елемената, *OpenWhisk* омогућава постављање акција заједно како би формирале секвенцу. Платформа се базира на архитектури покренутој догађајима, где се већина акција извршава као одговор на догађаје који се дешавају [36].

*OpenWhisk* подржава растућу листу језика попут *Go*, *Java*, *Node.JS*, *PHP*, *Python*, *Ruby*, *Scala* [74] и *Swift*. Такође постоји и експериментално окружење за *Denо* [75] у развоју. Уколико су потребни језици или библиотеке које тренутна „изван кутије“ извршна окружења не подржавају, могу се креирати и прилагодити сопствени извршни фајлови као ZIP акције које се извршавају на *Docker* извршном окружењу користећи *Docker Software Development Kit* (SDK). Када се напише функција, користи се *OpenWhisk CLI* (wsk) да би се погодила инстанца *Apache OpenWhisk* платформе и покренула акција у секундама [18].

Може се покренути акција десет хиљада пута у делићу секунде или једном недељно. Инстанце акције прилагођавају се потребама и оне нестају када више нису потребне [18].

Најлакши начин да се почне користити платформа је инсталација *Standalone* *OpenWhisk* стека. Ово је потпуно опремљени *OpenWhisk* стек који се изводи као процес имплементиран у *Java* програмском језику ради практичности. *Serverless* функције се изводе унутар *Docker* контејнера. На рачунару су неопходни унапред доступни *Docker*, *Java* и *Node.js*. За инсталацију стека, неопходно је извршити команду из листинга 3.1 [37].

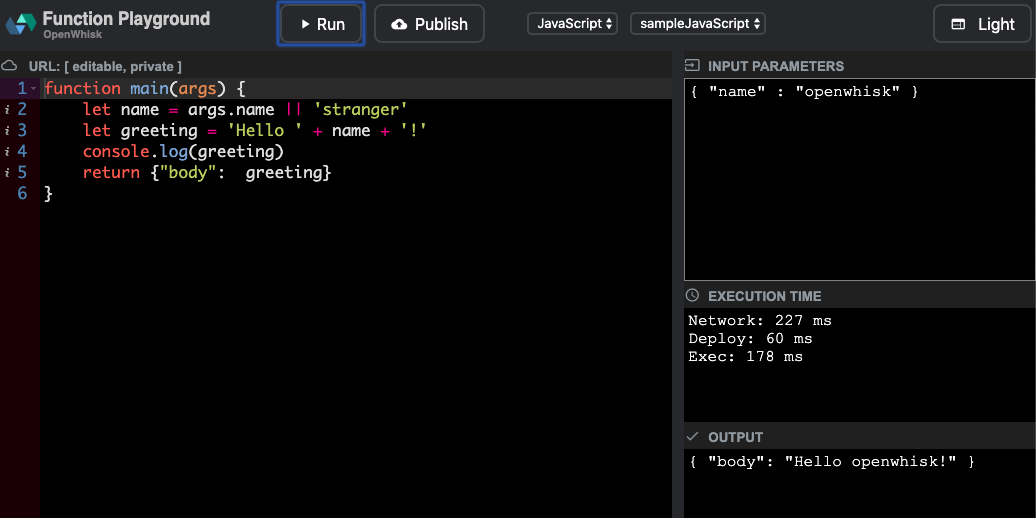
git clone https://github.com/apache/openwhisk.git

cd openwhisk

./gradlew core:standalone:bootRun

Листинг 3.1. Инсталација *OpenWhisk* стека

Када је *OpenWhisk* стек покренут, отвара се *Playground* кориснички интерфејс на интернет претраживачу. *Playground* омогућава да се директно из интернет претраживача креирају и покрећу функције, као што је приказано на слици 3.3. [37].



Слика 3.3. *Apache OpenWhisk Playground* кориснички интерфејс[[2]](#footnote-2)

Наредни део описује различите алате које можете користити за рад са *OpenWhisk* платформом, укључујући [19]:

* ***OpenWhisk* CLI (wsk)** – за једноставно креирање, покретање и управљање *OpenWhisk* ентитетима
* ***Whisk Deploy* (*wskdeploy*)** – алат који помаже да се имплементира и управља *OpenWhisk* пакетима, акцијама, окидачима, правилима и API-јима користећи манифест апликације.
* ***OpenWhisk* REST API** – након што се омогући *OpenWhisk* окружење, може се комуницирати са *web* или мобилним апликацијама путем REST API позива.
* ***OpenWhisk Clients*** – *OpenWhisk* пружа омотач око *OpenWhisk* API-ја у више различитих језика. Ови омотачи могу се увести и директно користити у бројним интегрисаним развојним окружењима.

*OpenWhsik CLI* (wsk) се инсталира тако што се преузме бинарни фајл са званичног *GitHub* [76] репозиторијума и премести у путању која је наведена у *PATH* системској променљивој. Ово ће омогућити позивање *OpenWhisk CLI* једноставним куцањем *wsk* у командној линији [19].

На идентичан начин се подешава и *Whisk Deploy* (*wskdeploy*), са којим након што је успешно подешен може се поставити *OpenWhisk* манифест фајл, једноставном командом:

* *wskdeploy -m example.yaml*

За евентуалну помоћ или претрагу могућих команди за *wsk* и *wskdeploy*, користе се наредне две команде:

* *wsk --help*
* *wskdeploy --help*

### 3.3.3. *Fission*

*Fission* [20] је радни оквир за *serverless* функције на *Kubernetes* систему. Омогућава писање кратких функција на бројним језицима и мапирање истих на HTTP захтеве или друге догађајне окидаче. Тренутно учитавање функција са једном командом. Нема контејнера за изградњу и нема управљања *Docker* регистрима [20].

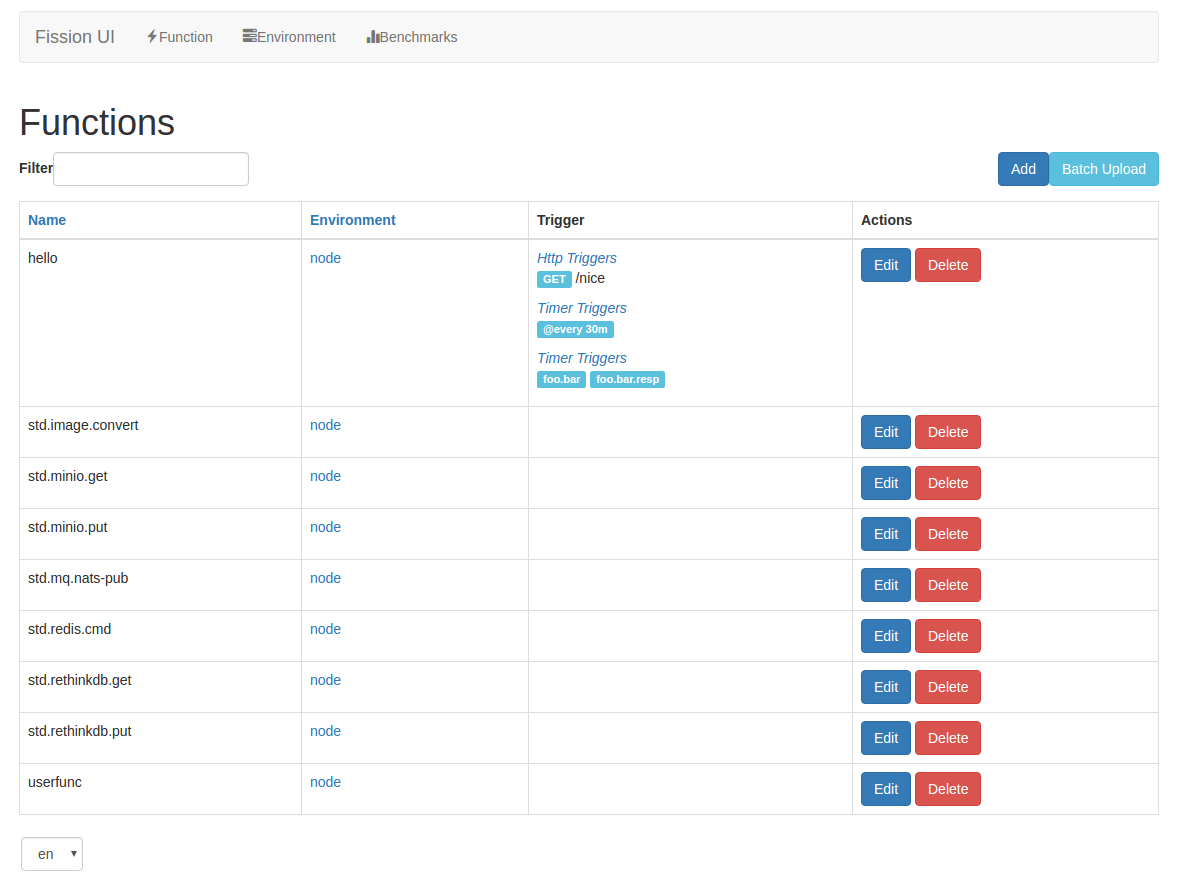
Платформа пружа многе бенефите, а неки од њих су [20]:

* **Фокус само на коду** – омогућава програмерима једноставно извршавање функција кода, истовремено аутоматизирајући задатке конфигурисања концепата *Kubernetes* микросервиса иза сцене. Само је потребно написати функције и *Fission* ће их покренути на *Kubernetes* систему.
* **Преносивост**: **могућност покретања услуге било где** – *Fission* је отвореног кода под *Apache* лиценцом. *Fission* ради на било ком *Kubernetes* кластеру: на персоналном рачунару, у било ком јавном рачунарству у облаку или у приватном центру података. Написане функције су флексибилне како би се могле имплементирати било где.
* **Проширивост и прилагодљивост** – платформа је проширива на било ком језику (*Python*, *NodeJS*, *Go*, *C#*, *PHP* су тренутно подржани). Претпостављени начин рада је апстракција контејнера, али може се изградити посебан контејнер ако је потребно.
* **Брзи старт** – платформа одржава конфигурабилни базен контејнера, тако да функције имају врло ниску латентност покретања, обично око 100 мили секунди.
* **Администрација и једноставност рада** – платформа тежи оперативној једноставности тако што директно укључује дневнике (енг. *logs*) у CLI, путем интеграције са *Fluentd* [77] платформом. Такође се интегрише са *Prometheus* системом за праћење метрика.
* ***Istio* сервис (интеграција мреже)** – платформа се интегрише са *Istio* [78] сервисом, који је платформа отвореног кода за повезивање, управљање и осигуравање микросервиса. Помоћу овог система корисници такође добијају могућност праћења употребе функција и праћења латентности захтева путем контролне табле.
* **Аутоматско скалирање функција** – платформа пружа скалирање функција на основу употребе процесора. У будућности ће такође бити подржане прилагођене метрике за скалирање функција. Могуће је поставити почетну и максималну употребу процесора за функцију, као и жељену употребу процесора при којој ће се покренути аутоматско скалирање.

Кључне компоненте на *Fission* платформи су:

* **Контролер** – садржи API за функције, HTTP окидаче, окружења, праћење догађаја у *Kubernetes* систему.
* **Окружење –** садржи контејнере специфичне за језик. Сваки контејнер окружења мора садржати HTTP сервер.
* **Извршилац –** креира *pod*-ове за функције. Постоје два типа извршиоца: *poolmgr* и *newdeploy*. *Poolmgr* креира скуп генеричких *pod*-ова одмах након што се окружење креира. Ови контејнери се називају топли контејнери. Када се функција позове, један од *pods*-ова се узима из окружења и користи за извршавање. Након одређеног периода неактивности *pod* се чисти и враћа у базен са осталим. *Poolmgr* је идеалан за ниску латентност, али се не може аутоматски скалирати. Са друге стране, *newdeploy* подржава аутоматско скалирање. Креира *Kubernetes deployment* заједно са сервисом и *Horizontal Pod Autoscaler*-ом за извршавање функције.
* **Усмеравач –** одговоран је за прослеђивање HTTP захтева на *pods* са функцијама
* **Kubewatcher** – прати *Kubernetes API* и позива функције повезане са праћењем догађаја.
* **Записивач** – одговоран је за прослеђивање дневника функција до централизованог места. Тренутно *Fission* чува дневнике на *InfluxDB* [79].

Платформа *Fission* такође поседује свој кориснички интерфејс који омогућава корисницима да посматрају и управљају функцијама, као што је приказано на слици 3.4. [38].



Слика 3.4. Fission кориснички интерфејс[[3]](#footnote-3)

Постоје два начина инсталације *Fission* платформе, један начин захтева употребу *Helm* руководиоца пакета, а други не [21].

Инсталација платформе употребом *Helm* пакета приказана је на листингу 3.2.

export FISSION\_NAMESPACE**=**"fission"

kubectl create namespace $FISSION\_NAMESPACE

kubectl create -k "github.com/fission/fission/crds/v1?ref=v1.19.0"

helm repo add fission-charts https://fission.github.io/fission-charts/

helm repo update

helm install --version v1.19.0 --namespace $FISSION\_NAMESPACE fission \

--set serviceType**=**NodePort,routerServiceType**=**NodePort \

fission-charts/fission-all

Листинг 3.2. Инсталација употребом *Helm* пакета

Инсталација платформе без употребе *Helm* пакета је приказана на листингу 3.3.

kubectl create -k "github.com/fission/fission/crds/v1?ref=v1.19.0"

export FISSION\_NAMESPACE**=**"fission"

kubectl create namespace $FISSION\_NAMESPACE

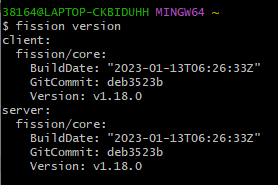
kubectl config set-context --current --namespace**=**$FISSION\_NAMESPACE

kubectl apply -f https://github.com/fission/fission/releases/download/v1.19.0/fission-all-v1.19.0.yaml

Листинг 3.3. Инсталација без употребе *Helm* пакета

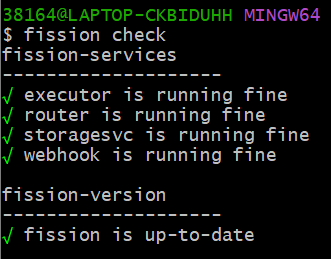
Након инсталације *Fission* платформе, остаје само инсталација *Fission CLI* алата који помаже при коришћењу платформе. *Fission CLI* се инсталира тако што се преузме бинарни фајл са званичног *GitHub* репозиторијума и премести у путању која је наведена у *PATH* системској променљивој.

Да би се проверило да ли је *Fission* успешно инсталиран, покретање команде *fission version* би требало да врати резултат у коме су клијентска и серверска верзија идентичне, као на слици 3.5.



Слика 3.5. Провера успешности инсталације *Fission* платформе

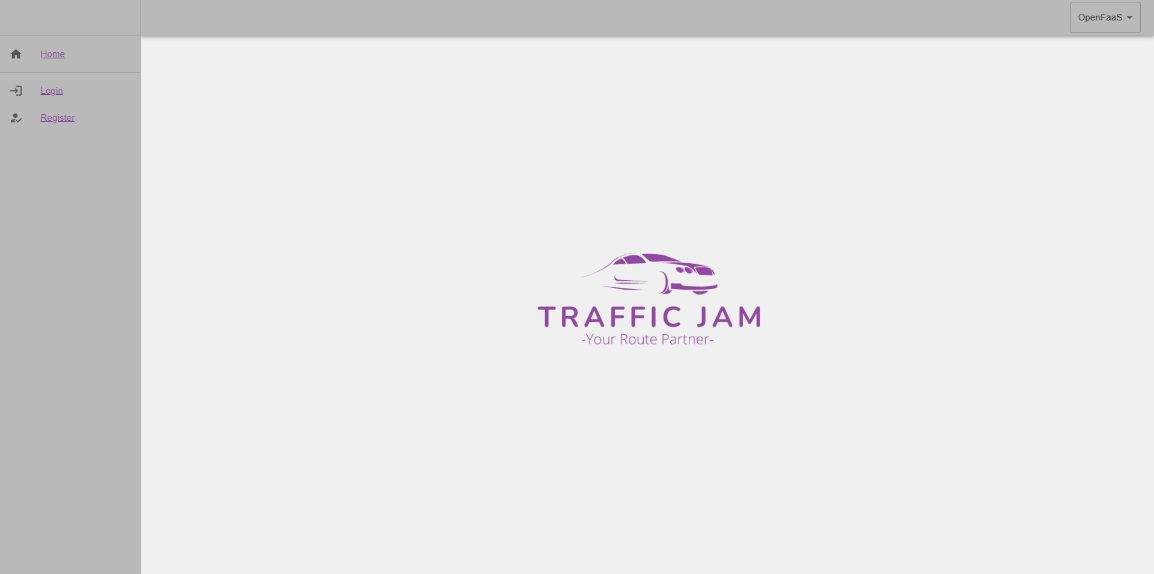
Да би се проверило да ли све *Fission* компоненте раде како треба, може се покренути команда *fission check* која би требала да врати резултат приказан на слици 3.6.



Слика 3.6. Провера исправности *Fission* платформе

# 4. Апликација за пријаву гужве у саобраћају

У савременим, пре свега градским окружењима појава гужве у саобраћају представља велики проблем. Дуготрајна задржавања у саобраћају доводе до стреса код возача и путника и у великој мери могу да утичу на ефикасност путовања, али и на квалитет самог живота. Циљ ове апликације јесте да праћењем и пријављивањем гужви у саобраћају пружи ефикасно решење за ове свакодневне проблеме. На слици 4.1. приказана је почетна страна ове *web* апликације.



Слика 4.1. Почетна страна апликације за пријаву гужве у саобраћају

## 4.1. Опис апликације

Главни циљеви апликације за пријаву гужве у саобраћају су:

* Олакшати корисницима пријаву гужве, као и осталих релевантних дешавања на путевима
* Пружити корисницима интуитивно окружење у виду мапе, како би се видели и означили различити маркери који означавају типове гужви као што су саобраћајне гужве, интервенције хитних служби или саобраћајне незгоде
* Омогућити корисницима да виде све пријављене маркере на мапи како би имали реалну слику о стању на путевима
* Обезбедити могућност администраторског управљања, које укључује и могућност уклањања корисника
* Унапредити координације и комуникације различитих актера на путевима

Функционалности апликације за пријаву гужве у саобраћају су:

* **Регистрација** – корисници се морају регистровати на систем користећи своје личне податке како би могли користити апликацију
* **Пријављивање** – корисници се морају пријавити на систем како би могли користити остале функционалности које апликација нуди
* **Маркери** – корисници апликације раде са различитим врстама маркера који представљају различите типове гужви, догађаја и хитних интервенција
* **Приказ локација маркера на мапи** – апликација приказује све пријављене гужве и догађаје на мапи како би корисници могли видети актуелне саобраћајне услове
* **Додавање новог маркера на мапама** – корисници могу додавати различите врсте маркера на мапи, како би означили различите типове гужви и догађаја
* **Приказ и уклањање корисника**– корисници са администраторским налозима могу видети све информације о осталим корисницима система и такође могу уклонити нежељене налоге

## 4.2. Архитектура система

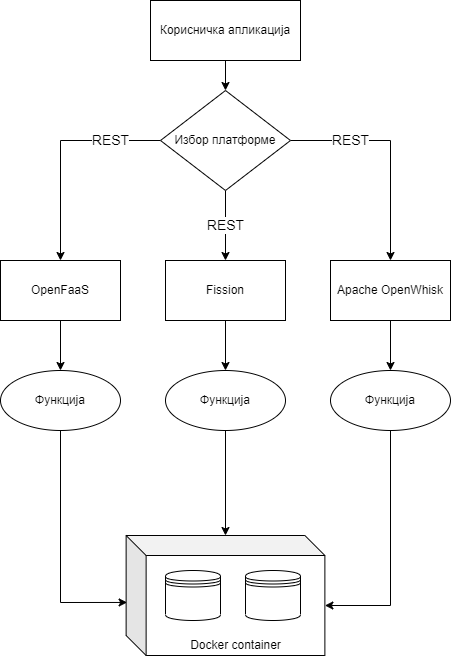
Апликација за пријаву гужве у саобраћају се састоји од две основне компоненте, а то су клијентска апликација и *serverless* платформе које имплементирају различите функционалности. Поред ове две основне компоненте, важно је споменути и базу података која се користи за складиштење података, као и протоколе који се користе за комуникацију.

Клијентска апликација представља интерфејс који корисници користе за комуникацију и приступ функционалностима. Корисници путем клијентске апликације приступају функционалностима апликације. Клијентска апликација омогућава корисницима унос података, преглед информација и интеракцију са сервером путем HTTP захтева.

*Serverless* платформе (*OpenFaaS, Fission, Apache OpenWhisk*) имплементирају пословну логику за апликацију и обрађују захтеве које им клијентска апликација шаље. Клијентска апликација комуницира са *serverless* платформама путем HTTP захтева. Платформе обрађују ове захтеве извршавајући одговарајуће функције. Свака од ових платформи имплементира цео скуп функционалности које апликација нуди. Ове платформе аутоматски управљају скалабилношћу, извршавају функције на пристигли захтев и омогућавају што ефикасније искоришћење постојећих ресурса.

База података складишти информације о корисницима, гужвама, догађајима и другим релевантним подацима. База је централизованог типа и користе је све три платформе.

На слици 4.2. приказана је општа архитектура апликације за пријаву гужве у саобраћају.



Слика 4.2. Општа архитектура система за пријаву гужве у саобраћају

## 4.3. Технологије и алати

Систем за пријаву гужве у саобраћају је платформа која се састоји од неколико кључних компоненти, које су имплементиране у различитим технологијама. Технологије коришћене за имплементацију описане су у овом делу рада.

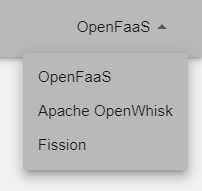
Као основни радни оквир за израду корисничке апликације коришћен је *React* [22]. У оквиру *React* радног оквира коришћене су бројне библиотеке које су олакшале израду клијентске апликације. Коришћена је библиотека *MUI Material* [23] која је омогућила креирање конзистентног и модерног изгледа апликације. Ова библиотека пружа велики скуп унапред дефинисаних компоненти, које су олакшале израду и побољшале изглед корисничке апликације. За интеграцију интерактивних мапа у корисничку апликацију коришћена је библиотека *React Leaflet* [24]. Ова библиотека пружила је алате за приказ и манипулацију мапа у *React* радном оквиру. Библиотека такође омогућава приказ и постављање различитих маркера на мапе. За комуникацију између корисничке апликације и *serverless* платформи коришћена је библиотека *Axios* [25]. *Axios* омогућава једноставно слање HTTP захтева, чиме је омогућена једноставна комуникација са платформама.

За имплементацију функционалности на све три *serverless* платформе коришћен је програмски језик *Python*. Програмски језик *Python* је веома флексибилан и подржан је на све три коришћене платформе. За обраду HTTP захтева са клијентске апликације коришћен је радни оквир за израду *web* апликација *Flask* [27]. *Flаsk* омогућава приступ телу захтева, као и читање параметара и тиме олакшава руковање HTTP захтевима и одговорима. За повезивање са *PostgresSQL* [28] базом података коришћена је библиотека *Psycopg2* [29]. Ова библиотека је дизајнирана за рад са *PostgresSQL* базом података и пружа ефикасан начин за повезивање са базом података и извршавање *Structured Query Language* (SQL) упита.

Систем чува све регистроване кориснике и релевантне информације у бази података. Ова база података се користи за складиштење локација гужви, информација о корисницима и података о типовима и озбиљности гужве. Као решење за складиштење података, коришћена је *PostgresSQL* база података. База се налази у *Docker* контејнеру, што омогућава лаку преносивост и изолацију. Све три *serverless* платформе користе исту базу података.

# 5. Имплементација система

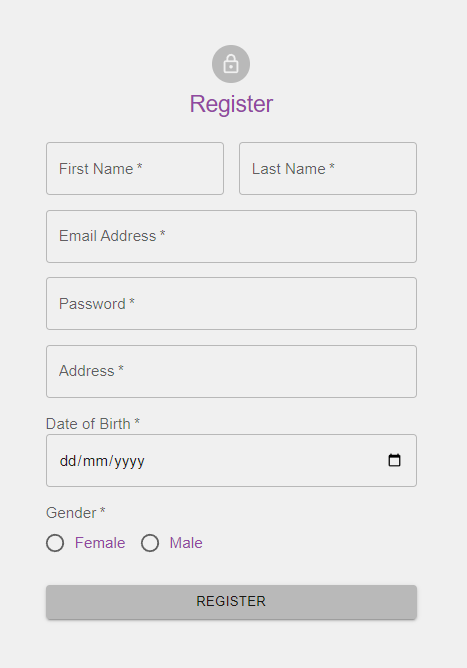
У овом делу биће размотрена и приказана имплементација система на све три платформе, кроз улоге које различити корисници имају у систему. Сви корисници у систему у свим моментима имају могућност промене *serverless* платформе са којом ће корисничка апликација комуницирати. Корисници могу одабрати платформу кликом на падајући мени који се налази у горњем десном углу и одабиром одговарајуће опције, као што је приказано на слици 5.1. Од тренутка када се одабере једна од понуђених платформи сви захтеви које корисник захтева упућују се искључиво на ту платформу. Ова функционалност омогућава корисницима да динамички бирају на којој платформи желе да изврше одређену функционалност. Ова функционалност омогућава оптимизацију перформанси и ресурса у зависности од конкретних захтева корисника.



Слика 5.1. Опција бирања *serverless* платформе

## 5.1. Нерегистровани корисник

Приликом отварања апликације нерегистровани корисници имају приступ почетној страници. На левој страни апликације поседују кориснички мени. У оквиру менија приметно је дугме које се означава као „*Register*“. Ово дугме је позиционирано поред дугмета за пријаву како би било јасно доступно корисницима који још немају креиран налог. Притиском на дугме за регистрацију корисници ће бити преусмерени на страницу за регистрацију. На овој страници се налази форма која захтева унос личних података од корисника, као што је приказано на слици 5.2.

Слика 5.2. Форма за регистрацију корисника

Након што корисник попуни сва потребна поља у регистрационој форми, кликом на дугме „*REGISTER*“ шаље се HTTP захтев на одабрану *serverless* платформу. *Serverless* платформе обрађују пристигли захтев и позивају одговарајућу функцију. Код за платформе *OpenFaaS* и *Fission* је идентичан и приказан је на листингу 5.1.

Платформа *Apache OpenWhisk* не садржи одговарајућу подршку за *Flask* радни оквир и код за ову и све остале функционалности је у одређеној мери измењен и прилагођен платформи. Имплементирана је библиотека која повезује FaaS слој са WSGI [30] интерфејсом. Имплементирана библиотека преводи непрерађен HTTP захтев са *Apache OpenWhisk* платформе у *Flask* захтев, а затим враћа одговор назад у *Apache OpenWhisk* одговор [31]. Имплементација овог мостa између *Apache OpenWhisk* платформе и *Flask* радног оквира је приказана на листингу 5.2.

def register\_user():

data = request.get\_json()

password = data.get('password')

firstname = data.get('firstname')

lastname = data.get('lastname')

email = data.get('email')

address = data.get('address')

birthdate = data.get('birthdate')

gender = data.get('gender')

if not firstname or not lastname or not password or not email or not address or not birthdate or not gender:

return json.dumps({'status': 400, 'message': 'Missing required data'})

hashed\_password = generate\_password\_hash(password, "sha256")

conn = psycopg2.connect(

host="host.docker.internal",

port="12107",

database="faas\_db",

user="postgres",

password="postgres"

)

cursor = conn.cursor()

cursor.execute(

"""

CREATE TABLE IF NOT EXISTS users (

id SERIAL PRIMARY KEY,

password VARCHAR(255) NOT NULL,

firstname VARCHAR(255) NOT NULL,

lastname VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL UNIQUE,

address VARCHAR(255),

birthdate DATE,

gender VARCHAR(10),

role VARCHAR(50) DEFAULT 'Client'

)

"""

)

conn.commit()

cursor.execute("SELECT COUNT(\*) FROM users WHERE email = %s", (email,))

user\_count = cursor.fetchone()[0]

if user\_count > 0:

conn.close()

return json.dumps({'status': 409, 'message': 'Email already exists'})

lastname, email, address, birthdate, gender) "

"VALUES (%s, %s, %s, %s, %s, %s, %s)",

(hashed\_password, firstname, lastname, email, address, birthdate, gender))

conn.commit()

conn.close()

return json.dumps({'status': 200, 'message': 'Registration successful'})

def main():

"""Handle registration request"""

if request.method == 'POST':

return register\_user()

else:

return json.dumps({'status': 405, 'message': 'Invalid request method'})

Листинг 5.1. Код за регистрацију корисника на *OpenFaaS* и *Fission* платформама

cursor.execute( "INSERT INTO users (password, firstname, lastname, email, address, birthdate, gender) "

"VALUES (%s, %s, %s, %s, %s, %s, %s)",

(hashed\_password, firstname, lastname, email, address, birthdate, gender))

conn.commit()

conn.close()

return json.dumps({'status': 200, 'message': 'Registration successful'})

def main():

"""Handle registration request"""

if request.method == 'POST':

return register\_user()

else:

return json.dumps({'status': 405, 'message': 'Invalid request method'})

def generate\_access\_token(email, role):

payload = {'email': email, 'role': role}

return jwt.encode(payload, "secret", algorithm="HS256")

def invoke(app,args):

environ = {

'REQUEST\_METHOD': args.get('\_\_ow\_method','GET').upper(),

}

if environ['REQUEST\_METHOD']=='POST':

raw = args.get('\_\_ow\_body')

decoded\_bytes = b64decode(raw)

decoded\_string = decoded\_bytes.decode('utf-8')

data = json.loads(decoded\_string)

email = data["email"]

password = data["password"]

if not email or not password:

return {

'statusCode': 400,

'headers': {'Content-Type': 'application/json'},

'body': json.dumps({'status': 400,'message': 'Missing required data'})

}

conn = psycopg2.connect(

host="host.docker.internal",

port="12107",

database="faas\_db",

user="postgres",

password="postgres"

)

cursor = conn.cursor()

cursor.execute("SELECT password, role FROM users WHERE email = %s", (email,))

user\_data = cursor.fetchone()

conn.close()

if not user\_data:

return {

'statusCode': 401,

'headers': {'Content-Type': 'application/json'},

'body': json.dumps({'status': 401, 'message': 'Invalid credentials'})

}

hashed\_password = user\_data[0]

role = user\_data[1]

if check\_password\_hash(hashed\_password, password):

access\_token = generate\_access\_token(email, role)

return {

'statusCode': 200,

'headers': {'Content-Type': 'application/json'},

'body': json.dumps({'status': 200, 'access\_token': access\_token})

}

return {

'statusCode': 401,

'headers': {'Content-Type': 'application/json'},

'body': json.dumps({'status': 401, 'message': 'Invalid credentials'})

}

else:

return {

'statusCode': 405,

'headers': {'Content-Type': 'application/json'},

'body': json.dumps({'status': 405, 'message': 'Invalid request method'})

}

Листинг 5.2. Код за регистрацију корисника на *Apache OpenWhisk* платформи

conn = psycopg2.connect(

host="host.docker.internal",

port="12107",

database="faas\_db",

user="postgres",

password="postgres"

)

cursor = conn.cursor()

cursor.execute("SELECT password, role FROM users WHERE email = %s", (email,))

user\_data = cursor.fetchone()

conn.close()

if not user\_data:

return {

'statusCode': 401,

'headers': {'Content-Type': 'application/json'},

'body': json.dumps({'status': 401, 'message': 'Invalid credentials'})

}

hashed\_password = user\_data[0]

role = user\_data[1]

if check\_password\_hash(hashed\_password, password):

access\_token = generate\_access\_token(email, role)

return {

'statusCode': 200,

'headers': {'Content-Type': 'application/json'},

'body': json.dumps({'status': 200, 'access\_token': access\_token})

}

return {

'statusCode': 401,

'headers': {'Content-Type': 'application/json'},

'body': json.dumps({'status': 401, 'message': 'Invalid credentials'})

}

else:

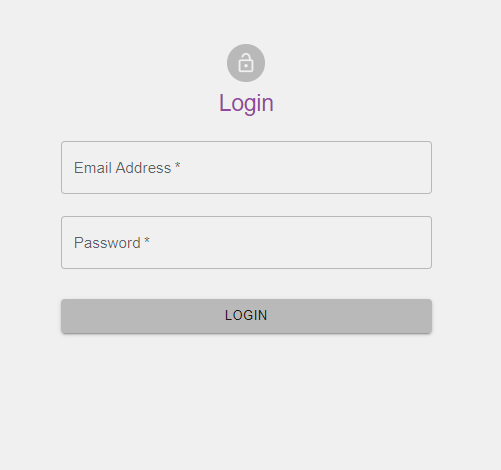
return {

'statusCode': 405,

'headers': {'Content-Type': 'application/json'},

'body': json.dumps({'status': 405, 'message': 'Invalid request method'})

}

У оквиру менија приметно је и дугме које се означава као „*Login*“. Притиском на дугме за пријаву корисници ће бити преусмерени на страницу за пријаву на систем. На овој страници се налази форма која захтева унос имејл адресе и лозинке од корисника, као што је приказано на слици 5.3.

Слика 5.3. Форма за пријаву корисника

Након што корисник попуни сва потребна поља у форми за пријаву, кликом на дугме „*LOGIN*“ шаље се HTTP захтев на одабрану *serverless* платформу. *Serverless* платформе обрађују пристигли захтев и позивају одговарајућу функцију. Све три платформе као одговор корисничкој апликацији враћају токен који садржи информације о пријављеном кориснику, а уколико пријава није могућа враћају информације о неуспешној пријави на систем. Код за платформе *OpenFaaS* и *Fission* је идентичан и приказан је на листингу 5.3.

Из разлога што платформа *Apache OpenWhisk* не садржи одговарајућу подршку за *Flask* радни оквир, обрада овог захтева је такође другачија у поређењу са остале две платформе. Главна разлика је у томе како се добављају подаци из пристиглог HTTP захтева и тај део кода је приказан на листингу 5.4.

Листинг 5.3. Код за пријаву корисника на *OpenFaaS* и *Fission* платформама

app = Flask(\_\_name\_\_)

def generate\_access\_token(email, role):

payload = {'email': email, 'role': role}

return jwt.encode(payload, "secret", algorithm="HS256")

def login\_user():

data = request.get\_json()

email = data.get('email')

password = data.get('password')

if not email or not password:

return json.dumps({'status': 400, 'message': 'Missing required data'})

conn = psycopg2.connect(

host="host.docker.internal",

port="12107",

database="faas\_db",

user="postgres",

password="postgres"

)

cursor = conn.cursor()

cursor.execute("SELECT password, role FROM users WHERE email = %s", (email,))

user\_data = cursor.fetchone()

conn.close()

if not user\_data:

return json.dumps({'status': 401, 'message': 'Invalid credentials'})

hashed\_password = user\_data[0]

role = user\_data[1]

if check\_password\_hash(hashed\_password, password):

access\_token = generate\_access\_token(email, role)

return json.dumps({'status': 200, 'access\_token': access\_token})

return json.dumps({'status': 401, 'message': 'Invalid credentials'})

def handle(req):

"""handle a login request"""

if request.method == 'POST':

return login\_user()

else:

return json.dumps({'status': 405, 'message': 'Invalid request method'})

Листинг 5.4. Код за обраду HTTP захтева на *Apache OpenWhisk* платформи

environ = {

'REQUEST\_METHOD': args.get('\_\_ow\_method','GET').upper(),

}

if environ['REQUEST\_METHOD']=='POST':

raw = args.get('\_\_ow\_body')

decoded\_bytes = b64decode(raw)

decoded\_string = decoded\_bytes.decode('utf-8')

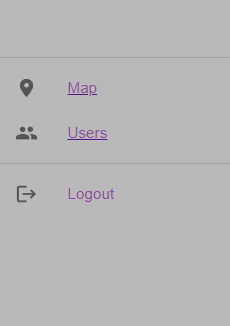
data = json.loads(decoded\_string)

email = data["email"]

password = data["password"]

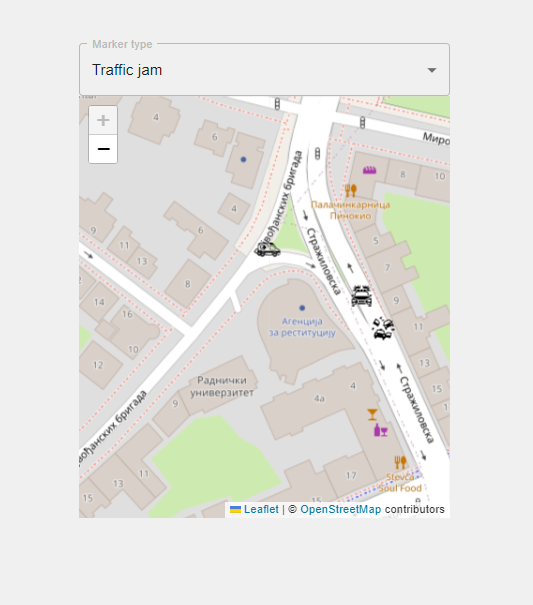
## 5.2. Регистровани корисник

Апликација омогућава две различите улоге у систему како би се омогућила контрола приступа различитим функционалностима. Корисници регистровани преко апликације добијају улогу обичног корисника, док су администратори директно додати у систем путем базе података. Администратор има приступ свим функционалностима као и обичан корисник уз могућност прегледа и управљања профилима свих корисника система. Мени регистрованог корисника приказан је на слици 5.4. Разлика између опција у менију обичног корисника и администратора је у томе што обичан корисник нема дугме означено са „*Users*“.

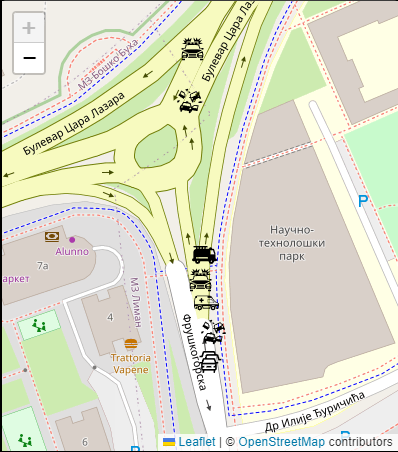


Слика 5.4. Приказ менија пријављеног корисника

У оквиру менија приметно је дугме које се означава као „*Map*“. Притиском на ово дугме корисници ће бити преусмерени на страницу за приказ мапе. На овој страници се налази мапа која пружа визуелни приказ актуелних догађаја у саобраћају путем интерактивне мапе, која је приказана на слици 5.5. Мапа је центрирана на тренутну географску локацију корисника, али корисник има и опцију кретања по мапи. На мапи се приказују различити маркери који означавају различиту врсту догађаја у саобраћају. Ови маркери се разликују по иконама и симболима како би корисницима било лакше препознати врсту догађаја, као што је приказано на слици 5.6.



Слика 5.5. Мапа са актуелним догађајима у саобраћају



Слика 5.6. Приказ различитих маркера на мапи

Након што корисник отвори страницу са приказом мапе, шаље се HTTP захтев на одабрану *serverless* платформу. *Serverless* платформе обрађују пристигли захтев и позивају одговарајућу функцију, која враћа списак свих маркера које је потребно приказати на мапи. Код за платформе *OpenFaaS* и *Fission* је идентичан и приказан је на листингу 5.5. Код на платформи *Apache OpenWhisk* се суштински не разликује од кода на остале две платформе, али је обрада пристиглог HTTP захтева прилагођена платформи, као што је већ раније приказано на листингу 5.4.

app = Flask(\_\_name\_\_)

class DecimalEncoder(json.JSONEncoder):

def default(self, o):

if isinstance(o, Decimal):

return float(o)

return super(DecimalEncoder, self).default(o)

def get\_markers():

conn = psycopg2.connect(

host="host.docker.internal",

port="12107",

database="faas\_db",

user="postgres",

password="postgres"

)

cursor = conn.cursor()

cursor.execute(

"""

CREATE TABLE IF NOT EXISTS markers (

id SERIAL PRIMARY KEY,

lat DECIMAL NOT NULL,

lng DECIMAL NOT NULL,

type VARCHAR(255) NOT NULL

)

"""

)

conn.commit()

cursor.execute("SELECT lat, lng, type FROM markers")

rows = cursor.fetchall() # Fetch all rows, not just one

conn.close()

markers\_data = []

for row in rows:

marker\_obj = {

'lat': row[0],

'lng': row[1],

'type': row[2]

}

markers\_data.append(marker\_obj)

return json.dumps({'status': 200, 'markers\_data': markers\_data}, cls=DecimalEncoder)

def handle(req):

"""handle a get-markers request"""

if request.method == 'GET':

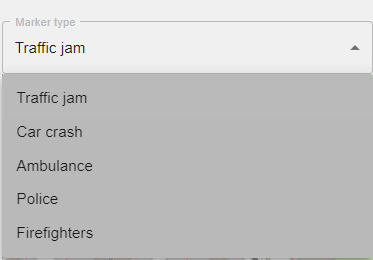
return get\_markers()

else:

return json.dumps({'status': 405, 'message': 'Invalid request method'})

Листинг 5.5. Код за добављање маркера на *OpenFaaS* и *Fission* платформама

Као што је приказано на слици 5.7. изнад мапе позиционирана је и компонента падајућег менија која омогућава корисницима да бирају између различитих опција које желе да пријаве у саобраћају.

 Слика 5.7. Опције одабира маркера

Корисници могу пријавити неки од следећих догађаја:

* Гужва у саобраћају
* Саобраћајна незгода
* Интервенција хитних служби
  + Хитне помоћи
  + Полиције
  + Ватрогасне службе

Када корисник изабере тип догађаја и кликне на мапу, корисничка апликација генерише и шаље HTTP захтев ка одабраној *serverless* платформи. У захтеву су садржане информације о типу догађаја и географској локацији догађаја. *Serverless* функција на одабраној платформи ће обрадити примљени захтев и додати маркер у базу података. Након што је маркер успешно додат, на корисничком интерфејсу се појављује порука о успешно пријављеном догађају и нови маркер се приказује на мапи. Код за платформе *OpenFaaS* и *Fission* је идентичан и приказан је на листингу 5.6. Код на платформи *Apache OpenWhisk* се суштински не разликује од кода на остале две платформе, али је обрада пристиглог HTTP захтева прилагођена платформи, као што је већ раније приказано на листингу 5.4.

app = Flask(\_\_name\_\_)

def add\_marker():

data = request.get\_json()

lat = data.get('lat')

lng = data.get('lng')

type = data.get('type')

if not lat or not lng or not type:

return json.dumps({'status': 400, 'message': 'Missing required data'})

conn = psycopg2.connect(

host="host.docker.internal",

port="12107",

database="faas\_db",

user="postgres",

password="postgres"

)

cursor = conn.cursor()

cursor.execute(

"""

CREATE TABLE IF NOT EXISTS markers (

id SERIAL PRIMARY KEY,

lat DECIMAL NOT NULL,

lng DECIMAL NOT NULL,

type VARCHAR(255) NOT NULL

)

"""

)

conn.commit()

cursor.execute( "INSERT INTO markers (lat, lng, type) "

"VALUES (%s, %s, %s)",

(lat, lng, type))

conn.commit()

conn.close()

return json.dumps({'status': 200, 'message': 'Marker successfully added'})

def handle(req):

"""Handle add marker request"""

if request.method == 'POST':

return add\_marker()

else:

return json.dumps({'status': 405, 'message': 'Invalid request method'})

def handle(req):

"""Handle add marker request"""

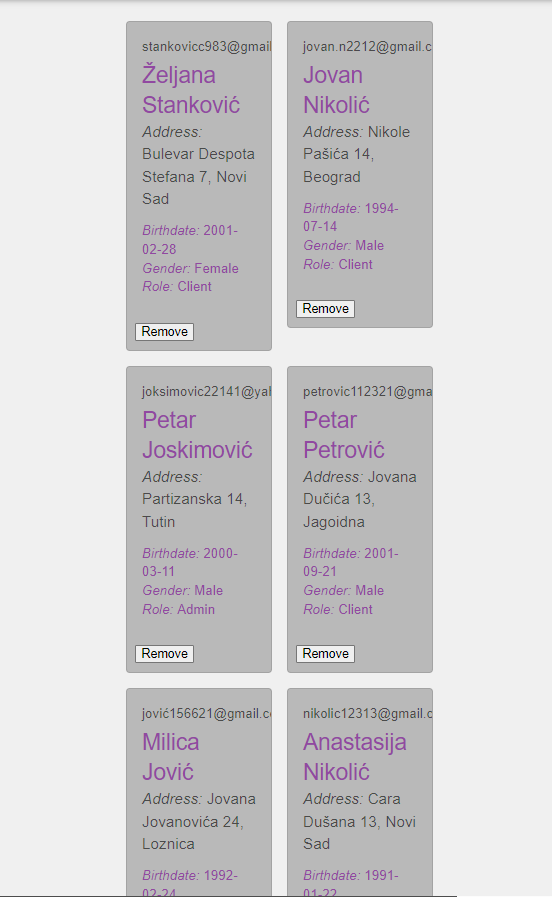
if request.method == 'POST':

return add\_marker()

else:

return json.dumps({'status': 405, 'message': 'Invalid request method'})

Листинг 5.6. Код за додавање маркера на *OpenFaaS* и *Fission* платформи

У оквиру менија корисника типа администратор приметно је и дугме које се означава као „*Users*“. Притиском на ово дугме администратори ће бити преусмерени на страницу за приказ информација о корисницима система. На овој страници се налази листа свих регистрованих корисника и основне информације о њима, као што је приказано на слици 5.8.

Слика 5.8. Приказ свих корисника система

Администратори имају овлашћење да уклоне кориснике који нису администратори на систему. Притиском на дугме „Remove“, корисничка апликација генерише и шаље HTTP захтев ка одабраној *serverless* платформи. У захтеву су садржане информације о кориснику чији налог је потребно уклонити. *Serverless* функција на одабраној платформи ће обрадити примљени захтев и уклонити корисника из базе података. Код за платформе *OpenFaaS* и *Fission* је идентичан и приказан је на листингу 5.6. Код на платформи *Apache OpenWhisk* се суштински не разликује од кода на остале две платформе, али је обрада пристиглог HTTP захтева прилагођена платформи, као што је већ раније приказано на листингу 5.7. уз додатак да платформа преузима при листингу 5.7. уз додатак да платформа преузима и информацију о кориснику из *query* параметра, као што је приказано на листингу 5.8.

Листинг 5.7. Код за уклањање корисничког налога на *OpenFaaS* и *Fission* платформама

app = Flask(\_\_name\_\_)

def delete\_user():

url = request.url

parsed\_url = urlparse(url)

email = parse\_qs(parsed\_url.query)['email'][0]

if not email:

return json.dumps({'status': 400, 'message': "You aren't provided email"})

conn = psycopg2.connect(

host="host.docker.internal",

port="12107",

database="faas\_db",

user="postgres",

password="postgres"

)

cursor = conn.cursor()

cursor.execute("SELECT COUNT(\*) FROM users WHERE email = %s", (email,))

conn.commit()

user\_count = cursor.fetchone()[0]

if user\_count == 0:

conn.close()

return json.dumps({'status': 409, 'message': "There isn't users with that email!"})

cursor.execute("DELETE FROM users WHERE email = %s",(email,))

conn.commit()

conn.close()

return json.dumps({'status': 200, 'message': "User successfully deleted!"})

def handle(req):

"""handle a get-markers request"""

if request.method == 'GET':

return delete\_user()

else:

return json.dumps({'status': 405, 'message': 'Invalid request method'})

Листинг 5.8. Код за преузимање *query* параметара нa *Apache OpenWhisk* платформи

def invoke(app, args):

headers = args.get('\_\_ow\_headers',{})

environ = {

'REQUEST\_METHOD': args.get('\_\_ow\_method','GET').upper(),

'QUERY\_STRING': args.get('\_\_ow\_query',None),

}

if environ['REQUEST\_METHOD'] == 'GET':

\_, email = environ['QUERY\_STRING'].split("=")

if not email:

return {

'statusCode': 400,

'headers': {'Content-Type': 'application/json'},

'body': json.dumps({'status': 400, 'message': "You aren't provided email"})

}

conn = psycopg2.connect(

host="host.docker.internal",

port="12107",

database="faas\_db",

user="postgres",

password="postgres"

)

cursor = conn.cursor()

cursor.execute("SELECT COUNT(\*) FROM users WHERE email = %s", (email,))

conn.commit()

user\_count = cursor.fetchone()[0]

# 6. Анализа *serverless* платформи отвореног кода

У овом поглављу анализиране су примећене предности и мане коришћених *serverless* платформи отвореног кода током процеса имплементације апликације за пријаву гужве у саобраћају. Такође пружен је и бољи увид у специфичности сваке платформе у контексту развоја и имплементације апликације, као и увид у перформансе и скалабилност платформи. Анализа ће омогућити и бољи увид у начин на који се свака платформа сналази у реалном окружењу и како утиче на процес развоја апликације. На крају поглавља сумирани су резултати анализе.

## 6.1. Анализа имплементације

### 6.1.1. *OpenFaaS*

Након што је платформа успешно инсталирана, осталo je само да се креирају *handler* датотеке и у њима имплементирају одговарајуће функције. Опционо потребно је дефинисати и датотеку са неопходним додатним библиотекама (*requirements.txt*). *Handler* датотеке обично садрже једну функцију која се извршава када пристигне одговарајући HTTP захтев.

У циљу олакшања процеса креирања нових функција, платформа нуди могућност коришћења шаблона који садрже основни код и структуру функције. Употреба шаблона у имплементацији апликације учинила је започињање развоја бржим и смањила потребу за поновним писањем основног кода. Шаблони имају уграђену подршку за *Python* програмски језик и *Flask* радни оквир, што је у великој мери олакшало развој апликације на овој платформи.

Након што су функције написане, један од начина њиховог постављања на *Docker* платформу јесте извршавање команди за креирање *Docker* слике и постављање слике на *Docker* складиште, као и команде за постављање функције у *Kubernetes* кластер. Ове команде су приказане на листингу 6.1.

faas-cli build login.yml

faas-cli push login.yml

faas-cli deploy login.yml

Листинг 6.1. Команде за постављање функције на *Docker* платформу

У циљу олакшања процеса активације и постављања имплементираних функција на *Docker* платформу, *OpenFaaS* платформа нуди аутоматизацију команди из листинга 6.1. и обједињује све команде у једну команду, која је приказана на листингу 6.2.

Листинг 6.2. Команда за аутоматизацију процеса постављања функција на *Docker* платформу

faas-cli up login.yml

Једна од значајних предности приликом имплементације апликације на платформи био је интуитивни и корисни *OpenFaaS Dashboard*, који је пружио бољи увид у све функције. Овај кориснички интерфејс омогућио је брз начин прегледа свих креираних функција, као и неке основне информације о њима. Ова карактеристика платформе, омогућила је да се на лак и брз начин дође до информација о функцијама које су креиране у прошлости и нису се користиле дуже време у току развоја.

Једна од кључних предности при имплементацији апликације била је изузетно квалитетна документација и редовно одржавање платформе. Квалитетна документација омогућила је да се веома брзо пронађу потребне информације. Велики број примера омогућио је боље разумевање саме платформе. Добра документација и редовно одржавање платформе, такође је омогућило и убрзање развојног циклуса и пружило могућност да се усредсреди на имплементацију пословне логике, а не на техничке изазове.

Приликом тестирања одређених нових, тек имплементираних функционалности, примећено је да је време чекања за одговор на HTTP захтев било дуже у поређењу са осталим платформама. Такође, примећено је да је у одређеном броју ситуација било неопходно рестартовати *minikube* контејнер како би се омогућила нормална обрада пристиглих HTTP захтева.

### 6.1.2. *Apache OpenWhisk*

Након што је платформа успешно инсталирана, остаје да се креирају различите датотеке, у зависности од програмског језика у коме се програмира и да се у њима имплементирају одговарајуће функције.

На овој платформи, изазов се појавио због недостатка директне подршке за *Flask* радни оквир, који је кључни део у имплементацији апликације за пријаву гужве у саобраћају. То је захтевало креирање посебног решења како би се омогућила интеграција *Flask* радног оквира са *Apache OpenWhisk* платформом. *Apache OpenWhisk* платформа је специфична по свом начину извршавања функција и то се није директно поклапало са начином на који *Flask* радни оквир функционише. Како би се интегрисала ова два елемента, изграђена је библиотека која служи као мост између *Apache OpenWhisk* платформе и *Flask* радног оквира. Ова библиотека има улогу да повеже FaaS слој *Apache OpenWhisk* платформе са WSGI интерфејсом који користи *Flask*. Ова библиотека има задатак да преводи непроцесуирани HTTP захтев са *Apache OpenWhisk* платформе у захтев који *Flask* разуме. Када *Flask* обради захтев и генерише одговор, библиотека га шаље натраг у *Apache OpenWhisk* одговор. Овај прилагођени приступ омогућио је коришћење

*Flask* радног оквира унутар *Apache OpenWhisk* платформе, иако директна подршка није постојала. Ово решење је захтевало више истраживања и били су потребни додатни напор и измене у постојећем коду, што је развојни циклус учинило дужим у поређењу са осталим платформама.

Након решеног проблема са употребом *Flask* радног оквира, рад са платформом је био поприлично интуитиван и једноставан процес. Платформа функције које су имплементиране и спремне за извршавање назива акцијама. Рад са акцијама је поприлично интуитиван и не захтева детаљно познавање платформе. Помоћу једноставних команди, могу се креирати и ажурирати акције, а платформа аутоматски обезбеђује окружење за извршавање акција. Пример команди за креирање и ажурирање акције, приказан је на листингу 6.3.

wsk action create login login.py –-web raw

wsk action update login login.py –-web raw

Листинг 6.3. Командe за рад са акцијама на платформи

Једна од већих предности ове платформе примећена након имплементиране функционалности је изузетно кратко време чекања на HTTP одговор након што је акција креирана и активирана. Платформа веома добро реагује на пристигле захтеве и омогућила је ефикасност у току развоја.

Током рада са платформом стекао се утисак да документација и подршка нису на завидном нивоу. Непотпуна документација доводила је до потешкоћа приликом конфигурације платформе, али и решавања проблема. Приликом појаве проблема са подршком за *Flask* радни оквир документација није била од велике помоћи. У том тренутку развоја, проблем није било могуће у потпуности идентификовати користећи документацију и материјал који пружа сама платформа. На крају је проблем идентификован и решен користећи друге изворe информација [31].

### 6.1.3. *Fission*

Након што је платформа успешно инсталирана, остало je само да се креирају датотеке са екстензијом одговарајућег програмског језика и у њима имплементирају одговарајуће функције. Опционо потребно је дефинисати и датотеку са неопходним додатним библиотекама (*requirements.txt*).

Креирање функција на *Fission* платформи је веома једноставан процес, што представља једну од кључних предности ове платформе. Након што је одрађена бизнис логика, неопходно је само дефинисати у ком раније дефинисаном окружењу ће функција бити покренута и у ком фајлу се налази сам код. Пример креирања функције приказан је на листингу 6.4.

fission fn create –name login -env python –code login.py

Листинг 6.4. Команда за креирање нове функције

Након што је нова функција успешно креирана, остало је још само креирање нове руте, како би се функција могла извршити слањем HTTP захтева. Пример креирања нове руте приказан је на листингу 6.5.

fission route create –method POST –url /login –function login

Листинг 6.5. Команда за креирање нове руте

Измена већ постојећих функција, такође је веома једноставан процес и такође се извршава у једној команди, као што је то приказано на листингу 6.6.

fission route create –method POST –url /login –function login

Листинг 6.6. Команда за измену постојеће функције

*Fission* платформа се истиче по изузетном квалитету своје документације и великој заједници која подржава нове кориснике платформе. Ова комбинација учинила је рад са платформом још продуктивнијим и лакшим.

Документација је добро структуирана и прегледна, и тако омогућава брз и лак проналазак информација које су потребне. *GitHub* складиште ове платформе пружа велики број практичних примера, што омогућава боље разумевање концепата и бржу практичну примену новог знања.

*Fission* платформа има веома активну заједницу корисника и развојних тимова који са корисницима деле своја знања, искуства и решења. Уколико корисник наиђе на проблем или има одређено питање везано за платформу, заједница пружа могућност постављања питања, на која ће искуснији корисници платформе поприлично брзо одговорити. Користећи *GitHub* складиште, корисници веома брзо добијају решења својих проблем, што омогућава продуктиван рад на платформи и спречава утрошак времена на тражењу решења ван оквира онога што платформа пружа.

Платформа пружа поприлично модеран кориснички интерфејс што је у великој мери олакшало сналажење са већим бројем функција и тако је убрзало развој апликације. Поред приказа свих функција на платформи, могу се видети и многе додатне информације о свакој од креираних функција.

Након интензивније употребе платформе примећено је као и код *OpenFaaS* платформе да је неопходно поновно покретање *minikube* контејнера, како би се омогућило добијање бржих одговора на пристигле HTTP захтеве.

## 6.2. Анализа скалабилности и перформанси

У овом делу рада, извршена је анализа перформанси и скалабилности *serverless* платформи, под различитим нивоима оптерећења. Перформансе и скалабилност су кључни аспекти сваке *serverless* платформе, јер директно утичу на способност система да ефикасно одговори на захтеве корисника у динамичком и променљивом окружењу. У наставку рада анализира се како су се платформе понашале у различитим сценаријима оптерећења.  
 Спецификације рачунара на којем су извршене све имплементације и анализе у оквиру овог рада су:

* Оперативни систем - *Windows 10 Home 64-bit*
* Процесор - *AMD Ryzen 5 Mobile 3500U*
* Радна меморија - *8GB Dual-Channel DDR4 – 1200MHZ*
* Матична плоча - *HUAWEI NBLK-WAX9X-PCB (FP5)*
* Графичка карта - *1024MB ATI AMD Radeon Vega 8 Graphics*

Све анализе и имплементације спроведене су искључиво на једном рачунару са наведеним спецификацијама, које су служиле као основно окружење за истраживање и експериментисање у оквиру овог рада, пружајући потребне ресурсе за извођење анализа и добијање резултата.

У циљу спровођења тестирања перформанси под различитим оптерећењима, коришћен је *Apache JMeter* [80], алат за тестирање оптерећења и перформанси *web* апликација. *Apache JMeter* омогућава симулацију великог броја корисника који истовремено приступају апликацији и шаљу захтеве. Након подешавања тестних сценарија у *Apache JMeter* алату, симулирани су различити сценарији оптерећења, укључујући ситуације са високим бројем истовремених захтева и тестирани су скалабилност и одзив сваке платформе. Алат је омогућио да се мери време одзива за сваки захтев, брзина извршавања функција, успешност послатих захтева и друге метрике перформанси.

У тестирању које је укључивало слање између 5 и 500 захтева послатих у исто време, примећене су значајне разлике у њиховом понашању.

При слању 5 захтева, све три платформе су постигле успешност од 100%, а време процесуирања било је релативно ниско, око 2 секунде. Ово указује на добру способност све три платформе да се носе са мањим оптерећењима.

Међутим, при повећању броја захтева на 25, *Apache OpenWhisk* се издвојио по брзини извршавања са временом од само 2,5 секунди за извршавање свих захтева, док су *OpenFaaS* и *Fission* показали дуже време процесуирања, од око 8 секунди. Ово сугерише да је *Apache OpenWhisk* ефикаснији при средњим оптерећењима.

Када је повећан број захтева на 50, *Fission* и *OpenFaaS* су и даље одржали успешност од 100%, али је време процесуирања повећано на 10 до 15 секунди. Насупрот томе, *Apache OpenWhisk* је доживео пад у успешности на 66%, али је задржао релативно брзо време извршавања свих захтева, од 6,5 секунди. Ови резултати указују на велике разлике у скалабилности између платформи под средњим оптерећењима.

При слању 100 захтева, *OpenFaaS* је и даље одржао високу успешност од 100% и време извршавања је било око 11 секунди. *Fission* је имао драстичан пад у успешности на само 15% и време процесуирања од око 15 секунди, *Apache OpenWhisk* је постигао средњу успешност од 44% и време извршавања је износило 13,6 секунди.

Коначно, при слању 500 захтева, све три платформе су се суочиле са значајним изазовима у успешности, при чему су *OpenFaaS*, *Fission* и *Apache OpenWhisk* имали успешност од око 16,6%, 1,8% и 9,6% респективно. Време извршавања је порасло на 13, 20 и 30 секунди респективно. Ови резултати указују на велике изазове са којима се сусрећу све три платформе под високим оптерећењем.

Важно је напоменути да су ово средње вредности добијене из више тестирања. На основу ових резултата може се закључити да свака од платформи има своје предности и мане у вези са скалабилношћу и перформансама. *Apache OpenWhisk* се издваја по брзини извршавања при неком средњем оптерећењу, док *OpenFaaS* и *Fission* одржавају високу успешност са нешто дужим временом извршавања, при средњем оптерећењу. Ипак, најстабилније понашање при различитим оптерећењима приказала је платформа *OpenFaaS.*

## 6.3. Сумирани резултати анализа

Након анализе имплементације и перформанси, као и упоређивања све три *serverless* платформе, добијени су одређени резултати. Сви закључци и резултати приказани су овој секцији, где ће бити сумирани у две табеле, како би се олакшао преглед предности и мана сваке од ових платформи. Прва табела ће обухватити предности платформи, док ће друга табела обухватити мане платформи. Ово омогућава јасно сагледавање кључних карактеристика сваке од платформи и олакшава доношење одлука о њиховом избору при различитим околностима.

У табели 6.1. приказане су кључне предности сваке платформе, док су у табели 6.2. приказане кључне мане сваке платформе.

|  |  |  |
| --- | --- | --- |
| *OpenFaaS* | *Apache OpenWhisk* | *Fission* |
| Брзо време извршавања функција | Брзо време извршавања функција | Једноставност |
| Употреба шаблона | Брзо реаговање на пристигле захтеве | Квалитетна документација |
| Кориснички интерфејс | Успешност при ниском оптерећењу | Активна корисничка заједница |
| Квалитетна документација | Брзина одговора на пристигле захтеве | Модеран кориснички интерфејс |
| Корисничка подршка |  | Успешност при ниском и средњем оптерећењу (до 50 истовремених захтева) |
| Активно одржавање платформе |  |  |
| Успешност при средњем оптерећењу (до 100 истовремених захтева) |  |  |

Табела 6.1. Предности анализираних платформи

|  |  |  |
| --- | --- | --- |
| *OpenFaaS* | *Apache OpenWhisk* | *Fission* |
| Успешност при већем оптерећењу (od 200 истовремених захтева) | Недостатак директне подршке за радне оквире (*Flask*) | Потреба за рестартовањем контејнера при раду |
| Потреба за рестартовањем контејнера при раду | Недостатак квалитетне документације и подршке | Успешност при већем оптерећењу (oд 100 истовремених захтева) |
|  | Захтева додатно истраживање и напор за решавање проблема са интеграцијом | Брзина процесуирања пристиглих захтева при већем оптерећењу |
|  | Успешност при средњем оптерећењу (од 50 истовремених захтева) |  |

Табела 6.2. Мане анализираних платформи

# 7. ЗАКЉУЧАК

У овом раду описана је и реализована апликација за пријаву гужве у саобраћају, са акцентом на имплементацију и упоређивање три платформе отвореног кода за *serverless* рачунарство: *OpenFaaS*, *Apache OpenWhisk* и *Fission*. Свака функционалност у оквиру апликације имплементирана је на све три платформе, што је пружило бољи увид у карактеристике, перформансе и могућности сваке од ових платформи. Функционалности су имплементиране у оквиру функција на самим платформама, које би се активирале када би пристигао HTTP захтев са клијентског дела апликације. Омогућено је праћење стања функција и ресурса на корисничким интерфејсима које платформе нуде. Апликација је омогућила динамичку измену коришћене платформе, како би се стекао бољи увид у могућности сваке од платформи.

Дат је увид у предности и мане *serverless* инфраструктуре. Поменути су алати коришћени за израду апликације и дат је бољи увид у сваки од њих. Описано је шта су то платформе отвореног кода за *serverless* рачунарство, као и њихове предности и мане. Описано је како функционише свака од наведених платформи и шта је све неопходно како би се оне инсталирале. Детаљно су представљене предности и мане сваке од платформи и посебно су акцентоване јединствене могућности које неке платформе пружају. Дат је увид у то како стећи више детаља о стању платформи и функција написаних у оквиру њих.

У даљем току рада приказана је имплементација апликације за пријаву гужве у саобраћају. Дат је детаљан увид у функционалности које апликација нуди и архитектуру система. Описано је како одабрати платформу на којој ће одређена функционалност бити извршена. Описане су улоге у систему, као и могућности како регистрованих, тако и нерегистрованих корисника. Приказан је код *serverless* функција које су извршаване на платформама на пристигле захтеве.

У завршном делу рада анализиране су информације добијене при имплементацији апликације за пријаву гужве у саобраћају. Извршена су тестирања и анализирани резултати при сценаријима који укључују већи број пристиглих захтева на платформама. Сви добијени закључци из анализа сумирани су у табелама.

У даљем истраживању могло би да се фокусира на анализу перформанси при још већем броју захтева. Унапређења у корисничкој апликацији могла би да се изврше додавањем нових типова корисника, као и додавањем нових овлашћења администраторима. Такође би могла да се врше тестирања при раду са већом количином података и са већим бројем корисника, али и да се обрати пажња на сигурносне аспекте.

## ЛИТЕРАТУРА

[1] Serverless computing | IBM Topics <https://www.ibm.com/topics/serverless> (приступ: 2023-05-25).

[2] What are some serverless use cases? | RedHat Topics <https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless> (приступ: 2023-05-25).

[3] Docker <https://www.docker.com/> (приступ: 2023-05-26).

[4] Developing with Docker | Docker <https://www.docker.com/why-docker/> (приступ: 2023-05-26).

[5] What container | Docker <https://www.docker.com/resources/what-container/> (приступ: 2023-05-26).

[6] Kubernetes <https://kubernetes.io/> (приступ: 2023-05-26).

[7] Kubernetes vs. Docker <https://www.atlassian.com/microservices/microservices-architecture/kubernetes-vs-docker> (приступ: 2023-05-26).

[8] Minikube <https://minikube.sigs.k8s.io/docs/> (приступ: 2023-05-26).

[9] OpenFaaS <https://www.openfaas.com/> (приступ: 2023-05-27).

[10] OpenFaaS | Docs <https://docs.openfaas.com/> (приступ: 2023-05-27).

[11] faas-cli <https://github.com/openfaas/faas-cli> (приступ: 2023-05-27).

[12] Van Eyk, Erwin, et al. "Serverless is more: From paas to present cloud computing." IEEE Internet Computing 22.5 (2018): 8-17.

[13] Baldini, Ioana, et al. "Serverless computing: Current trends and open problems." Research advances in cloud computing (2017): 1-20.

[14] Nupponen, Jussi, and Davide Taibi. "Serverless: What it is, what to do and what not to do." 2020 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE, 2020.

[15] Heron, Michael, Vicki L. Hanson, and Ian Ricketts. "Open source and accessibility: advantages and limitations." Journal of interaction Science 1.1 (2013): 1-10.

[16] Trappler, Thomas. "Is there such a thing as free software? The pros and cons of open source software." EDUCAUSE quarterly 32.2 (2009): 10.

[17] Đurković, Jovica, Vuk Vuković, and Lazar Raković. "Open source approach in software development—Advantages and disadvantages." Manag Inforation Syst 3 (2008): 029-33.

[18] Apache OpenWhisk <https://openwhisk.apache.org/> (приступ: 2023-05-29).

[19] Apache OpenWhisk Docs<https://openwhisk.apache.org/documentation.html>

(приступ: 2023-05-29).

[20] Fission <https://fission.io/> (приступ: 2023-05-30).

[21] Fission | Docs <https://fission.io/docs/> (приступ: 2023-05-30).

[22] React <https://react.dev/> (приступ: 2023-08-21).

[23] Material UI <https://mui.com/material-ui/> (приступ: 2023-08-21).

[24] React Leaflet <https://react-leaflet.js.org/> (приступ: 2023-08-21).

[25] Axios <https://axios-http.com/> (приступ: 2023-08-21).

[26] Python <https://www.python.org/> (приступ: 2023-08-21).

[27] Flask <https://flask.palletsprojects.com/en/2.3.x/> (приступ: 2023-08-21).

[28] PostgresSQL <https://www.postgresql.org/> (приступ: 2023-08-21).

[29] Psycopg2 <https://www.psycopg.org/> (приступ: 2023-08-21).

[30] WSGI <https://wsgi.readthedocs.io/en/latest/> (приступ: 2023-08-22).

[31] A library that bridges the FaaS layer to the WSGI <https://github.com/alexmilowski/flask-openwhisk> (приступ: 2023-08-22).

[32] Shafiei, Hossein, Ahmad Khonsari, and Payam Mousavi. "Serverless computing: a survey of opportunities, challenges, and applications." *ACM Computing Surveys* 54.11s (2022): 1-32.

[33] Rad, Babak Bashari, Harrison John Bhatti, and Mohammad Ahmadi. "An introduction to docker and analysis of its performance." *International Journal of Computer Science and Network Security (IJCSNS)* 17.3 (2017): 228.

[34] Kubernetes, T. "Kubernetes." *Kubernetes. Retrieved May* 24 (2019): 2019.

[35] Rossi, Fabiana, Simone Falvo, and Valeria Cardellini. "GOFS: Geo-distributed scheduling in OpenFaaS." 2021 IEEE Symposium on Computers and Communications (ISCC). IEEE, 2021.

[36] Quevedo, Sebastián, et al. "Evaluating apache openwhisk-faas." 2019 IEEE fourth ecuador technical chapters meeting (ETCM). IEEE, 2019.

[37] Apache OpenWhisk | GitHub <https://github.com/apache/openwhisk> (приступ: 2023-08-29).

[38] Fission UI <https://github.com/fission/fission-ui> (приступ: 2023-08-23).

[39] Javascript <https://www.javascript.com/> (приступ: 2023-08-24).

[40] Java <https://www.java.com/en/> (приступ: 2023-08-29).

[41] Go <https://go.dev/> (приступ: 2023-07-02).

[42] C# <https://learn.microsoft.com/en-us/dotnet/csharp/> (приступ: 2023-08-24).

[43] Swift <https://www.swift.org/> (приступ: 2023-08-24).

[44] What is FaaS (Function as a service)? <https://www.ibm.com/topics/faas#:~:text=the%20next%20step-,What%20is%20FaaS%3F,building%20and%20launching%20microservices%20applications>. (приступ: 2023-08-24).

[45] AWS Lambda <https://aws.amazon.com/lambda/> (приступ: 2023-08-24).

[46] Linux <https://www.linux.org/> (приступ: 2023-08-24).

[47] Windows <https://www.microsoft.com/en-us/windows?r=1> (приступ: 2023-08-24).

[48] Ubuntu <https://ubuntu.com/> (приступ: 2023-08-24).

[49] What is SQL (Structured Query Language)? <https://aws.amazon.com/what-is/sql/#:~:text=Structured%20query%20language%20(SQL)%20is,information%20in%20a%20relational%20database>. (приступ: 2023-08-24).

[50] containerd <https://containerd.io/> (приступ: 2023-08-24).

[51] CRIO <https://kubernetes.io/docs/concepts/architecture/cri/> (приступ: 2023-08-25).

[52] YAML <https://yaml.org/> (приступ: 2023-08-25).

[53] What is a Pod? <https://kubernetes.io/docs/concepts/workloads/pods/> (приступ: 2023-08-25).

[54] MacOS <https://support.apple.com/macos> (приступ: 2023-08-25).

[55] Oracle VM VirtualBox <https://www.virtualbox.org/> (приступ: 2023-08-25).

[56] Vmwarе <https://www.vmware.com/> (приступ: 2023-08-25).

[57] Kernel Virtual Machine <https://www.linux-kvm.org/page/Main_Page> (приступ: 2023-08-25).

[58] REST

<https://restfulapi.net/> (приступ: 2023-08-25).

[59] Prometheus <https://prometheus.io/> (приступ: 2023-08-25).

[60] Ruby <https://www.ruby-lang.org/en/> (приступ: 2023-08-25).

[61] Node.js <https://nodejs.org/en> (приступ: 2023-08-25).

[62] PHP <https://www.php.net/> (приступ: 2023-08-25).

[63] Express.js <https://expressjs.com/> (приступ: 2023-08-25).

[64] Eclipse Vert.x <https://vertx.io/> (приступ: 2023-08-25).

[65] ASP.NET Core <https://learn.microsoft.com/en-us/aspnet/core/> (приступ: 2023-08-25).

[66] FastAPI <https://fastapi.tiangolo.com/> (приступ: 2023-08-25).

[67] Django <https://www.djangoproject.com/> (приступ: 2023-08-25).

[68] Helm <https://helm.sh/> (приступ: 2023-08-26).

[69] Flux CLI <https://fluxcd.io/flux/cmd/> (приступ: 2023-08-26).

[70] Argo CD <https://argo-cd.readthedocs.io/> (приступ: 2023-08-26).

[71] Grafana <https://grafana.com/> (приступ: 2023-08-26).

[72] Apache Mesos <https://mesos.apache.org/> (приступ: 2023-08-26).

[73] Red Hat OpenShift <https://www.redhat.com/en/technologies/cloud-computing/openshift> (приступ: 2023-08-26)

[74] Scala <https://www.scala-lang.org/> (приступ: 2023-08-27).

[75] Deno <https://deno.land/> (приступ: 2023-08-27).

[76] Github <https://github.com/> (приступ: 2023-08-27).

[77] Fluentd <https://www.fluentd.org/> (приступ: 2023-08-28).

[78] Istio <https://istio.io/> (приступ: 2023-08-28).

[79] InfluxDB <https://www.influxdata.com/> (приступ: 2023-08-28).

[80] Apache JMeter <https://jmeter.apache.org/> (приступ: 2023-09-09)

[81] Fox, Geoffrey C., et al. "Status of serverless computing and function-as-a- service (faas) in industry and research." arXiv preprint

arXiv:1708.08028 (2017).

# 

# БИОГРАФИЈА

Дарко Селаковић рођен је 11. јануара 2001. године у Ужицу. Основну школу „Богосав Јанковић“ у Кремнима завршио је 2015. године. Након тога уписује Техничку школу у Ужицу, коју завршава 2019. године. Исте године уписује се на Факултет техничких наука, смер Рачунарство и аутоматика. Школске 2021/22. се опредељује за усмерење Примењене рачунарске науке и информатика, након чега се школске 2022/23. опредељује за модул Интелигентни системи. Положио је све испите предвиђене планом и програмом.

# КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

|  |  |
| --- | --- |
| Редни број, **РБР**: |  |
| Идентификациони број, **ИБР**: |  |
| Тип документације, **ТД**: | монографска публикација |
| Тип записа, **ТЗ**: | текстуални штампани документ |
| Врста рада, **ВР**: | дипломски рад |
| Аутор, **АУ**: | Дарко Селаковић |
| Ментор, **МН**: | проф. др Горан Сладић, ФТН Нови Сад |
| Наслов рада, **НР**: | Поређење платформи отвореног кода за serverless рачунарство |
| Језик публикације,**ЈП**: | српски |
| Језик извода, **ЈИ**: | српски / енглески |
| Земља публиковања, **ЗП**: | Србија |
| Уже географско подручје, **УГП**: | Војводина |
| Година, **ГО**: | 2023 |
| Издавач, **ИЗ**: | ауторски репринт |
| Место и адреса, **МА**: | Нови Сад, Факултет техничких наука,  Трг Доситеја Обрадовића 6 |
| Физички опис рада, **ФО**: | бр. поглавља 7/ страница 71/ цитата 1/ табела 2/ слика 19/ графикона / прилога / листинга 18 |
| Научна област, **НО**: | Електротехничко и рачунарско инжењерство |
| Научна дисциплина, **НД**: | Софтверско инжењерство |
| Предметна одредница/кључне речи, **ПО**: | serverless рачунарство, веб апликација, OpenFaaS, Apache OpenWhisk, Fission |
| **УДК** |  |
| Чува се у, **ЧУ**: | Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад |
| Важна напомена, **ВН**: |  |
| Извод, **ИЗ**: | Описана је изградња апликације за пријаву гужве у саобраћају на три платформе – OpenFaas, Apache OpenWhisk и Fission. Анализиранe су предности и мане сва три решења. Приказани су имплементациони детаљи од значаја. |
| Датум прихватања теме, **ДП**: |  |
| Датум одбране, **ДО**: |  |
| Чланови комисије, **КО**: |  |
| председник |  |
| члан |  |
| ментор | проф. др Горан Сладић, ФТН Нови Сад |
| Потпис ментора | |

# KEY WORDS DOCUMENTATION

|  |  |
| --- | --- |
| Accession number, **ANO**: |  |
| Identification number, **INO**: |  |
| Document type, **DT**: | monographic publication |
| Type of record, **TR**: | textual material |
| Contents code, **CC**: | BSc thesis |
| Author, **AU**: | Darko Selaković |
| Mentor, **MN**: | Goran Sladić, PhD, full prof., FTN Novi Sad |
| Title, **TI**: | Comparison of open source platforms for serverless computing |
| Language of text, **LT**: | serbian |
| Language of abstract, **LA**: | serbian / english |
| Country of publication, **CP**: | Serbia |
| Locality of publication, **LP**: | Vojvodina |
| Publication year, **PY**: | 2023 |
| Publisher, **PB**: | author’s reprint |
| Publication place, **PP**: | Novi Sad, Faculty of Technical Sciences, Square Dositej Obradović 6 |
| Physical description, **PD**: | no. of chapters 7/ pages 71/ quotes 1/ tables 2/ pictures 19/ graphs / appendix / listings 18 |
| Scientific field, **SF**: | Electrical and computer engineering |
| Scientific discipline, **ND**: | Software engineering |
| Subject / Keywords, **S/KW**: | serverless computing, web application, OpenFaaS, OpenWhisk, Fission |
| **UDC** |  |
| Holding data, **HD**: | Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad |
| Note, **N**: |  |
| Abstract, **AB**: | The thesis describes the development of an application for reporting traffic congestion on three platforms - OpenFaas, Apache OpenWhisk, and Fission. The advantages and disadvantages of all three solutions are analyzed. Important implementation details are presented. |
| Accepted by sci. board on, **ASB**: |  |
| Defended on, **DE**: |  |
| Defense board, **DB**: |  |
| president |  |
| member |  |
| mentor | Goran Sladić, PhD, full prof., FTN Novi Sad |
| Mentor’s signature | |

1. Слика преузета са званичне Docker странице <https://www.docker.com/resources/what-container/> [5] [↑](#footnote-ref-1)
2. Слика преузета са званичног *Github* складишта <https://github.com/apache/openwhisk> [37] [↑](#footnote-ref-2)
3. Слика преузета са званичног *Github* складишта <https://github.com/fission/fission-ui> [38] [↑](#footnote-ref-3)