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

Дарко Селаковић

**Поређење open source платформи за serverless computing**

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: |  |
| Руководилац студијског програма: |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: |  | Број индекса: |  |
| Област: |  | | |
| Ментор: |  | | |
| 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:**

|  |
| --- |
|  |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
|  |

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

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

САДРЖАЈ

[1.УВОД 7](#_Toc143719562)

[2. Примена serverless инфраструктуре 9](#_Toc143719563)

[2.1. Предности serverless инфраструктуре 9](#_Toc143719564)

[2.2. Мане serverless инфраструктуре 10](#_Toc143719565)

[2.3. Коришћени алати 12](#_Toc143719566)

[2.3.1. Docker 12](#_Toc143719567)

[2.3.1.1. Docker Container 12](#_Toc143719568)

[2.3.2. Kubernetes 13](#_Toc143719569)

[2.3.2.1. Pod 13](#_Toc143719570)

[2.3.2.2. Minikube 14](#_Toc143719571)

[3. Оpen source платформе 17](#_Toc143719572)

[3.1. Предности open source платформи 17](#_Toc143719573)

[3.2. Мане open source платформи 18](#_Toc143719574)

[3.3. Коришћене платформе 18](#_Toc143719575)

[3.3.1. OpenFaaS 18](#_Toc143719576)

[3.3.1.1. Инсталација 19](#_Toc143719577)

[3.3.2. Apache OpenWhisk 21](#_Toc143719578)

[3.3.3. Fission 24](#_Toc143719579)

[4. Апликација за пријаву гужве у саобраћају 29](#_Toc143719580)

[4.1. Опис апликације 29](#_Toc143719581)

[4.2. Архитектура система 30](#_Toc143719582)

[4.3. Технологије и алати 31](#_Toc143719583)

[5. Имплементација система 33](#_Toc143719584)

[5.1. Нерегистровани корисник 33](#_Toc143719585)

[5.2. Регистровани корисник 39](#_Toc143719586)

[6. ЗАКЉУЧАК 47](#_Toc143719587)

[ЛИТЕРАТУРА 49](#_Toc143719588)

[БИОГРАФИЈА 53](#_Toc143719589)

[КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА 55](#_Toc143719590)

[KEY WORDS DOCUMENTATION 57](#_Toc143719591)

# УВОД

Овај рад је фокусиран на поређење *open source* платформи за *serverless computing*. Последњих година *open source* платформе 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.

*Serverless* инфраструктура 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 провајдер рачунарства у облаку. Провајдер рачунарства у облаку се брине за све остало, обезбеђујући инфраструктуру потребну за покретање кода и динамички скалирајући инфраструктуру према захтевим. Провајдер 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].

Мотивација за настанак *open source* платформи за *serverless computing* 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.

У наставку рада биће анализирано неколико оваквих платформи, истражене карактеристике, функционалности и перформансе.

# 2. Примена *serverless* инфраструктуре

*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. Задатак као што је обрада велике количине долазних сликовних датотека, који може бити покренут ретко, али мора бити спреман када у тренутку стигне велика серија слика. Или задатак попут праћења долазних промена у бази података, а затим примењивање низа функција, попут провере промена према стандардима квалитета или аутоматског превођења. *Serverless* апликације се такође користе у случајевима који укључују долазне токове података, chat ботове, заказане задатке или пословну логику. Неке друге честе употребе *serverless* технологије су *back-end* Application Programming Interface *(*API) *web* апликације, аутоматизација пословних процеса, *serverless* веб сајтови и интеграција између више система [2].

*Severless* *computing* 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 Amazon, Google и 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 computing* настоји постићи аутоматско скалирање уз пружање повољних услуга [32].

## 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 традиционалном моделу рачунарства у облаку, корисник је одговоран за одабир и имплементацију конкретних ресурса. Да би се избегло оптерећење корисника опцијама, опсег опција је обично ограничен на велике, вишефункционалне врсте ресурса (нпр. контејнери). Апликације ретко одговарају тим ресурсима, а како би се смањио терет који проистиче из коришћења великих ресурса опште намене, апликације су грубо подељене. Насупрот томе, *serverless computing* значи да су апликације ситно подељене, што значи да пружалац услуга рачунарства у облаку може прецизније ускладити апстрактну потражњу ресурса са стварним системским ресурсима [12].
* **Већи увид и контрола –** у традиционалном моделу, корисник је одговоран за имплементацију, праћење и друге оперативне задатке везане за животни циклус грубо подељених апликација. Међутим, многи корисници услуга рачунарства у облаку немају потребно стручно знање. Осим тога, оператори немају контекст па морају доносити одлуке о аутоматском скалирању без прецизних профила или увида из имплементираних апликација. Са *serverless* инфраструктуром, повећана одговорност оператора пружа више увида и контроле. Оператори бирају, имплементирају и обезбеђују ресурсе, такође имплементирају и контролишу праћење коришћења ресурса, интензитета оптерећења и понашања апликација. Могу аутоматски скалирати или мигрирати апликацију. Они могу профилисати и моделирати ситне услуге које чине serverless апликацију, понудити ове информације корисницима и побољшати одлуке донете уз помоћ ових увида [12].
* **Ситно склаирање –** у традиционалном моделу, апликације се састоје од великих, вишенаменских виртуалних машина са временом испоруке од неколико минута. Ове виртуелне машине делују као црне кутије и стога је тешко моделирати и предвидети њихово понашање за операторе. Иако су апликације обично успорене само једним ресурсом на одређеном делу апликације, оператори морају скалирати само целу апликацију како би решили проблем. Елиминација неких од ових проблема је могућа, али захтева поновно осмишљање апликације, обично као микросервисе апликације. Ефективност тога зависи веома од стручности корисника – већина корисника не може имати користи од тога. Са *serverless* инфраструктуром, оператор може боље скалирати појединачне, ситне услуге или функције, користећи дубоке увиде [12].
* **Програмски језици** – функционалности на платформама ове врсте могу бити имплементиране у великом броју подржаних језика, међу којима су и JavaScript [39], Java [40], Python [26], Go [41], C# [42] и Swift [43]. Већина платформи подржава више од једног програмског језика. Неке од тих платформи такође подржавају механизме проширивости за код написан на било ком језику [13].
* **Извршавање кода** – Платформе теже да олакшају процес извршавања кода колико год је могуће. Обично је довољно да програмери само обезбеде датотеку са изворним кодом функције. Осим тога, постоји много опција где се код може упаковати као архива са више датотека унутар ње или као Docker слика са бинарним кодом. Такође, могућности за груписање функција и додељивање верзија истим су корисне, али ретке [13].
* **Остали бенефити –** поред набројаних предности *serverless* инфраструктура пружа још много бенефита који иду у прилог њеном усвајању. Премештање са капиталних трошкова на оперативне трошкове, тачније усклађивање трошкова са стварним пословним процесима је један од бенефита. Такође, независне услуге омогућавају тимовима да одаберу праве алате за одређени случај употребе, без утицаја на друге делове система и организације. Висок ниво апстракције омогућава пројектантима софтвера да брже итерирају кроз ове дистрибуиране системе, смањујући потребу за обимним стручним знањем из дистрибуираних система [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 клијенте Function as a service (FaaS) [44] модел који нуди платформа може бити превише ограничавајући за неке апликације. На пример, платформа можда не подржава најновију верзију Python програмског језика или одређене библиотеке можда неће бити доступне [13].
* **Ограничења за добављаче услуга** – данас постоји потреба за управљањем проблемима као што су животни циклус корисничких функција, скалабилност и толеранција грешака. Ово такође значи да програмери морају пажљиво да разумеју како се платформа понаша и да дизајнирају апликацију наспрам њених способности [13].
* **Асинхрони позиви** – асинхрони позиви ка *serverless* функцијама и између њих повећавају сложеност система. Обично се удаљени API позиви прате моделом захтева и одговора и лакше се имплементирају са синхроним позивима *serverless* функција. Усвојено решење је то да се користе синхрони позиви када год је то могуће. Користити интеграцију путем реда порука како би се обавестио позивалац о успеху операције уколико се користе асинхрони позиви. Међутим, асинхрони позиви су одрживо решење за једнократне послове попут покретања дуготрајног процеса израде резервне копије [14].
* **Позиви функција другим функцијама** – комплексно откривање грешака и слаба изолација функционалности карактеришу ову акцију. Јављају се додатни трошкови ако се функције позивају синхроно јер морамо платити за истовремено извршавање две функције. Усвојено решење је избегавање позивања функције из неке друге функције или спајање функција, када је то могуће [14].
* **Дељени код између функција** – може доћи до пуцања постојећих *serverless* функција које зависе од дељеног кода који је промењен. Постоји ризик да се достигне ограничење величине слике (500MB у већини платформи) и дуже време покретања (што је слика већа, дуже траје покретање). Усвојено решење је да се пишу независне и одвојене функције. Користити чисту архитектуру и зависности дељеног кода само путем добро дефинисаних и тестираних интерфејса [14].
* **Употреба превише библиотека** – повећање простора који користе библиотеке повећава ризик од достизања ограничења величине слике и продужава време покретања. Усвојено решење је да се увозе само библиотеке које су заиста потребне, избегавајући преоптерећење система. Једно могуће решење које се користи у AWS Lambda [45] окружењу, ако време загревања није битно, је учитавање библиотеке при покретању из спољашњег складишта у привремени директоријум [14].
* **Превише функција** – проблем настаје креирањем нових функција у случајевима када је могуће употебити већ постојеће. Неактивне *serverless* фунцкије не коштају ништа, па постоји искушење да се уместо измењивања постојеће функционалности према промењеним захтевима, стварају нове функције. Проблем представљају смањена одрживост и нижа разумљивост система. Усвојено решење је да се пажљиво разматра да ли постоји потреба за креирањем нове функције. Груписати функције у микросервисе, тако да друге услуге виде само интерфејс микросервиса уместо детаљне имплементације појединачних функција [14].
* **Трансформација старог кода** – велики изазов програмерима може представљати трансформација старог код који се не извршава на *serverless* платформама на нови код који ће бити извршаван на овим платформама. Економска вредност постојећег кода представља огромну инвестицију у бројним сатима програмирања и исправљања софтвера. Због тога, један од најважнијих проблема може бити у којој мери се постојећи стари код може аутоматски или полуаутоматски разложити на мање делове, како би се искористиле нове економске предности које нуди *serverless* инфраструктура [13].

## 2.3. Коришћени алати

За подешавање окружења и имплементацију *open source* платформи за *serverless* *computing* важни су алати који се за то користе. Ови алати играју кључну улогу у омогућавању ефикасног развоја, тестирању и скалирању *serverless* апликација. Поред одговрајућих 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 *open source* платформа 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 виртуализоване 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].

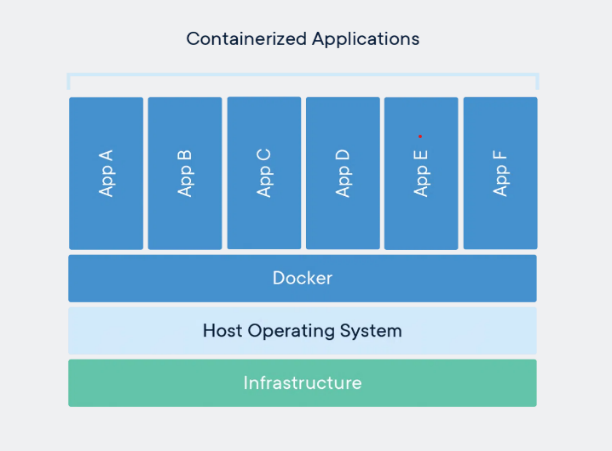
Docker уклања понављајуће и замарајуће задатке конфигурације и користи се током целог животног циклуса развоја за брз, једноставан и преносив развој апликација за *desktop* и зарачунарство у облаку. Свеобухватна Docker платформа од почетка до краја укључује графичке корисничке интерфејсе, командне линије 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 ваш 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 контејнер је стандардна јединица 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. 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 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 container инфраструктура [5]

### 2.3.2. Kubernetes

Kubernetes [6], такође познат као K8s, је *open source* систем 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 за вас 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 резличитих 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 инсталирате 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 ће распоређивати 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]

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.

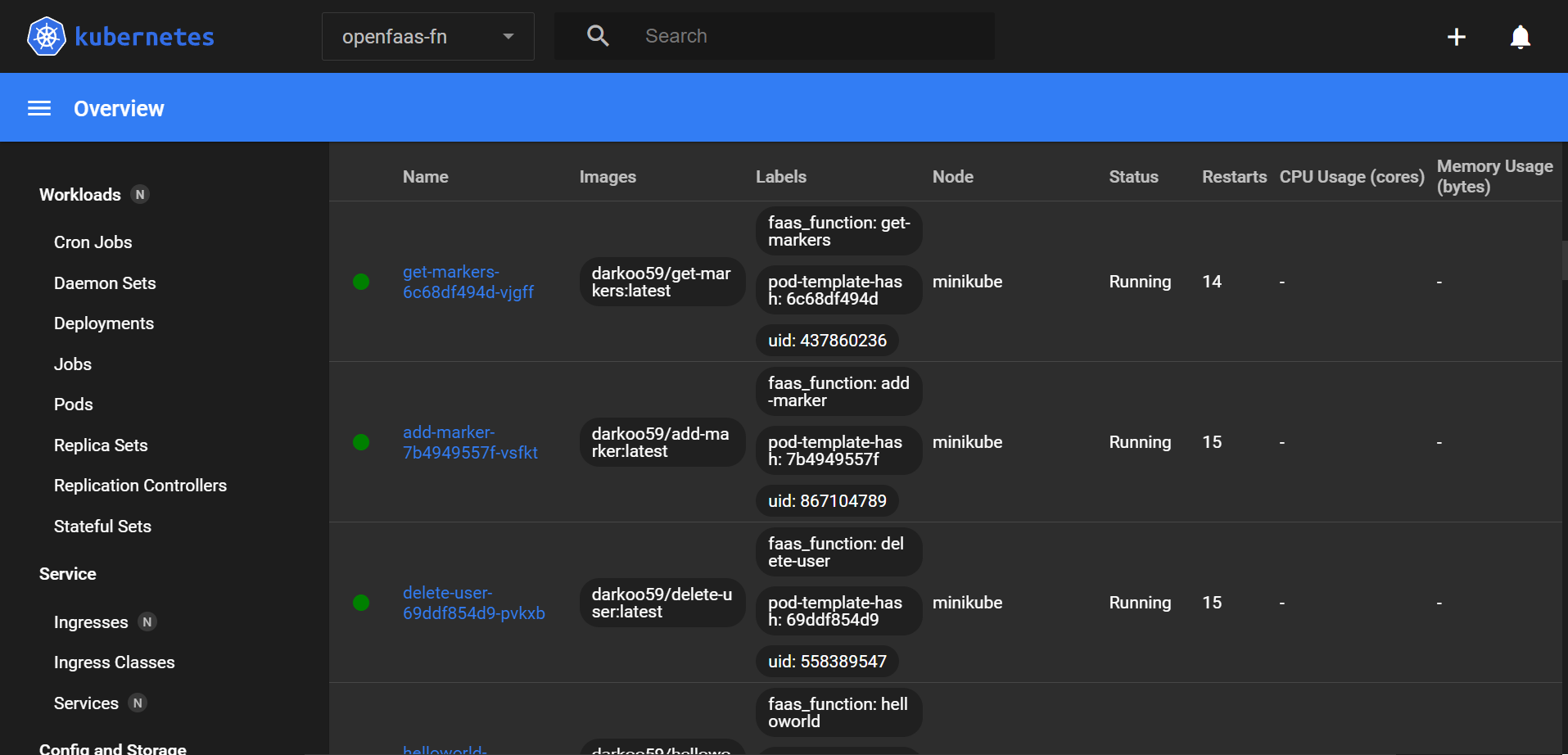
Када се pod креира, Kubernetes ће га надгледати и аутoматски га поново покренути ако се процес прекине. Поред тога, Kubernetes се може конфигурисати да покуша да се повеже са контејнером преко мреже како би утврдио да ли је pod спреман (readinessProbe) и да ли је још увек активан (livenessProbe) [34].

#### 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 виртуализациони 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. ***Оpen source* платформе**

Постоји велики број *open source* платформи за *serverless computing*, 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 *open source* платформи

Ван контекста истраживања, постоји много убедљивих разлога за аутора да објави софтвер као *open source*. Међу њима су очекивања да ће понуда бесплатног кода повећати тржишни део и веровање да ће изградња базе развоја око алата повећати дугорочну одрживост, као и 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 у смислу пристрасности према даваоцу лиценце или примаоцу лиценце. Кратки и јасни услови олакшавају усклађивање и нема потребе пратити употребу лиценце [16].
* **Флексибилност** - приступ изворном коду омогућава прилагођавање, ограничено само интерним ресурсима. Доступност изворног кода олакшава другима да врше прилагођавања и често цела заједница може имати користи од таквих промена. Коришћење ових софтвера пружа више тачака одлучивања, не само у софтверу, већ и у подршци и хардверу, а више избора значи већу флексибилност [16].
* **Смањење грешака** – многе мале, али константне измене у коду, као и тестирање у којем учествује много стручњака, доприносе смањењу грешака. Управо таква широко распрострањена тестирања су врло ефикасна у развоју *open source* софтвера. Логично је да многи корисници могу открити више грешака. Заједнице комуницирају са различитим развојним тимовима, што повећава могућност да бар једној особи грешка буде очигледна. Осим тога, одговорнима за проналажење грешака пружа се могућност избора, што је додатна мотивација за лоцирање проблема [17].
* **Мотивација** – програмери прихватају искуство које заједница нуди и стичу афирмацију зависну од остварених резултата. Гледано са овог становишта, извор мотивације налази се у пројектима и изазовима који произилазе из пројекта [17].
* **Одсуство притиска код развојног тима** – планирање оваквог пројекта је неформално. Не постоје конкретни планови и визије, већ дугорочни циљ, односно напредак пројекта. То није проблем. Напротив, нејасно дефинисани циљеви и одсуство рокова могу побољшати продуктивност развојног тима. Многи овакви пројекти су врло продуктивни, посебно они који се заснивају на добровољном раду и учесници нису обавезни да раде пуно радно време. Одсутан је притисак наметнут планирањем [17].

## 3.2. Мане *open source* платформи

Поред присутних бројних предности, као и код било које технологије, *open source* платформе нису без својих 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** – када говоримо о контроли квлаитета, *open source* пројекти се ослањају само на критике заједнице. Остале врсте тестирања су неуобичајене, посебно у области дизајна. Веома је важно правилно развијати пројекте, као и тестирати њихов квалитет. Критике заједнице понекад нису способне да идентификују озбиљне грешке у архитектури. Осим тога, оне су такође неефикасне у откривању неприметних грешака. То су грешке које се ретко јављају и које није лако идентификовати током процеса тестирања кода. Осим тога, ефикасност критика заједнице може бити нејасна. Може се десити да једна особа пронађе грешку, али многи други ће провести много времена тражећи је због недостатка информација [17].
* **Документација** – већина *open source* пројеката улаже велике напоре у писање адекватне документације. Често је лакше обезбедити ресурсе за одржавање информационог система него обезбедити документацију која садржи информације о томе. Због тога, већина знања и вештина о *open source* пројектима остаје скривена и није забележена. То отежава учење о пројектима и непотребно одузима време [17].
* **Сигурност** – код је отворен, што значи да сви корисници могу да га виде, што може довести до бржег откривања и решавања проблема. То такође значи да коду могу приступити и злонамерни корисници, који могу да га прегледају и пронађу потенцијалне слабости за злоупотребу. У сваком случају, корисник увек може да прегледа и утврди стварни ниво безбедности пре усвајања производа [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 [9]:

* Платформа за преносиве функције – извршавање функције на било којој локалној или инфраструктури рачунарства у ваздуху
* Могућност писања функција на било ком језику и запакивања у Docker/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 Proof of Concept (PoC) без икаквих трошкова

Ова платформа је једна од највише коришћених *open-source serverless* платформи, захваљујући својој лаганој и проширивој архитектури. 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 REST [58] АPI-ја и прикупља метрике на нивоу функција путем Prometheus-a [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. Постоји 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 има велику заједницу *open source* сарадника и корисника. Платформа је написана у језику Go и добродошли су доприноси корисника и заједнице. То може значити пружање повратних информација кроз тестирање функционалности, предлагање побољшања или укључивање у одржавање скоро 50 пројеката. Језгро OpenFaaS платформе је независни *open source* пројекат који је оригинално креирао Alex Ellis 2016. године. Сада се пројекат развија и обликује од стране све веће заједнице сарадника и корисника [10].

#### 3.3.1.1. Инсталација

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

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

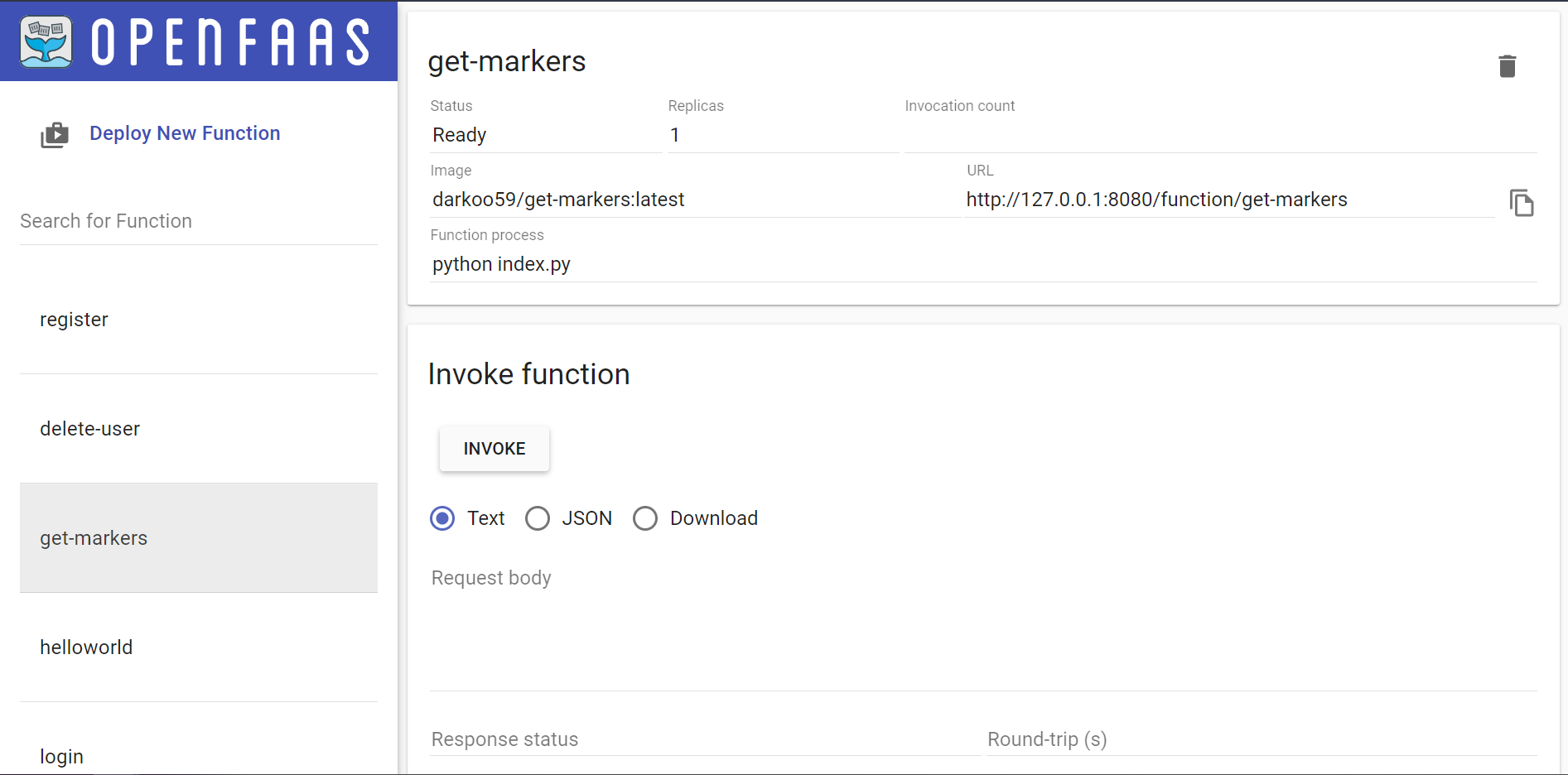
Официјални CLI за OpenFaaS је faas-cli. CLI алат се може користити за изградњу и имплементацију функција на OpenFaaS платформи. Могуће је изградити OpenFaaS функције користећи подржане језичке шаблоне. То значи да само треба написати фајл за руковођење (као што је handler.py/handler.js), а CLI алат ће 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 инсталирате faas-cli, можете почети креирати и имплементирати функције путем следећих наредби [10]:

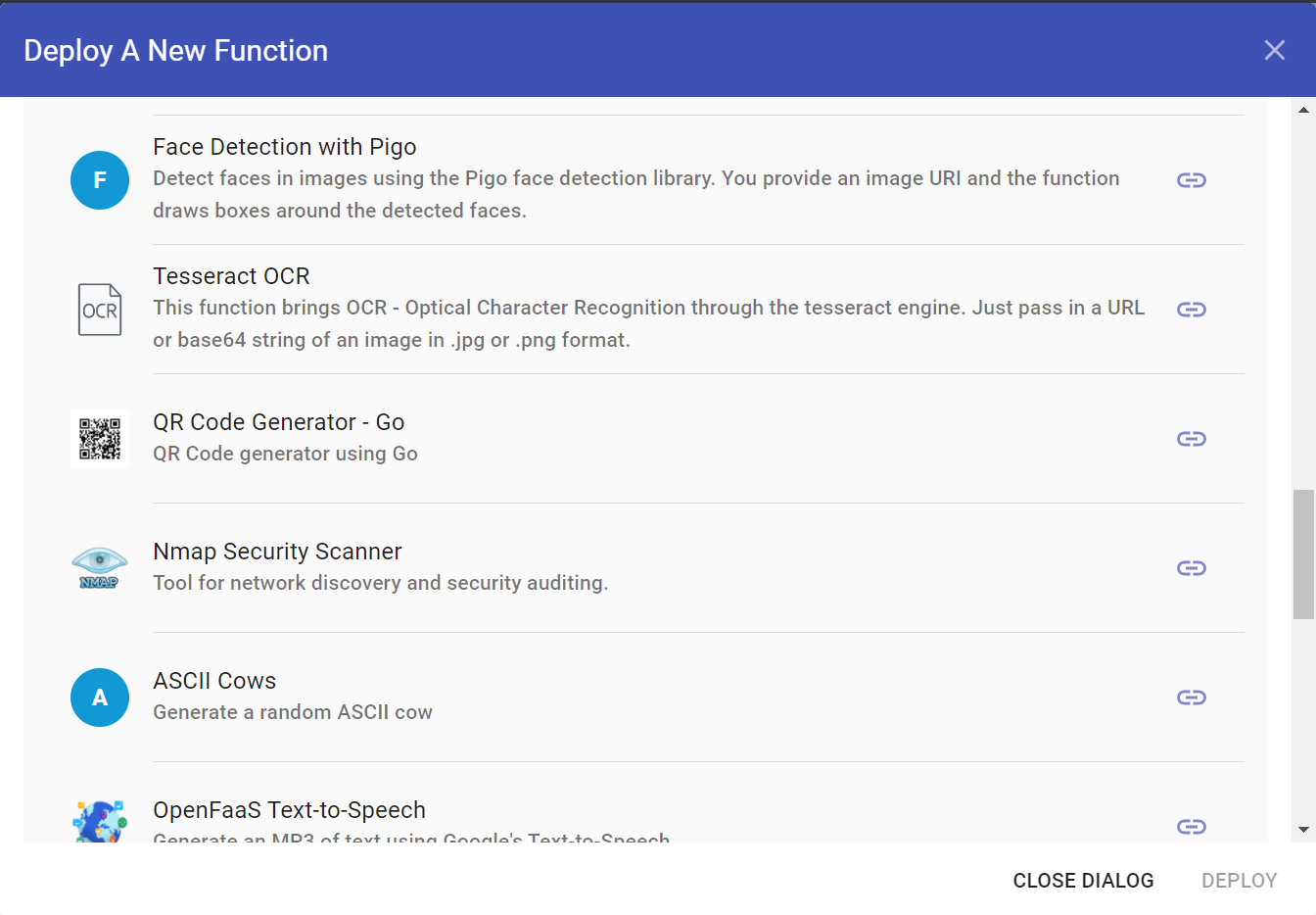
* faas-cli build – изградња слике у локалној Docker библиотеци
* faas-cli push – слање те слике на удаљени регистар контејнера
* faas-cli deploy – имплементација функција на кластеру
* faas-cli up – аутоматизује све горенаведене кораке у једној наредби

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

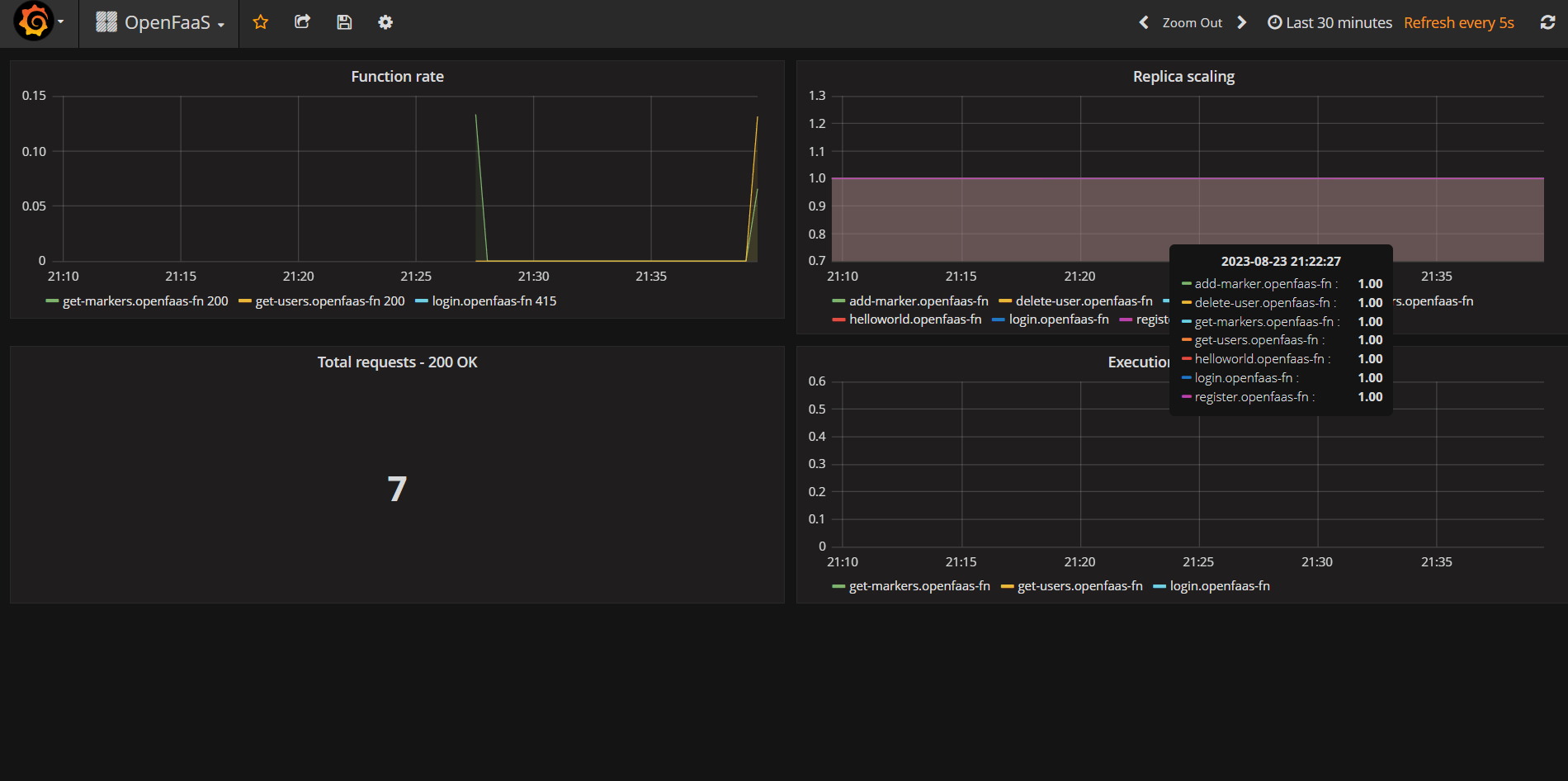
На слици 3.1. приказан је изглед OpenFaaS Dashboard корисничког интерфејса.



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 *open source* алата као што је Grafana [72], као што је то приказано на слици 3.3.

Слика 3.2. Приказ праћења метрика путем алата Grafana

### 3.3.2. Apache OpenWhisk

Apache OpenWhisk је *open source*, 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 прогрмаски 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 Hyper Transfer Protocol (HTTP) захтева. Пројекат укључује Command Line Interface (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].

Платформа омогућава да радите са оним што знате и волите. OpenWhisk подржава растућу листу језика попут Go, Java, Node.JS, PHP, Python, Ruby, Scala [74] и Swift. Такође постоји и експериментално окружење за Deno [75] у развоју. Уколико су потребни језици или библиотеке које тренутна “изван кутије” извршна окружења не подржавају, могу се креирати и прилагодити споствени 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].

#### 3.3.2.1. Инсталација

Најлакши начин да се почне користити платформа је инсталација *Standalone* OpenWhisk стека. Ово је потпуно опремљени OpenWhisk стек који се изводи као Java процес 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].

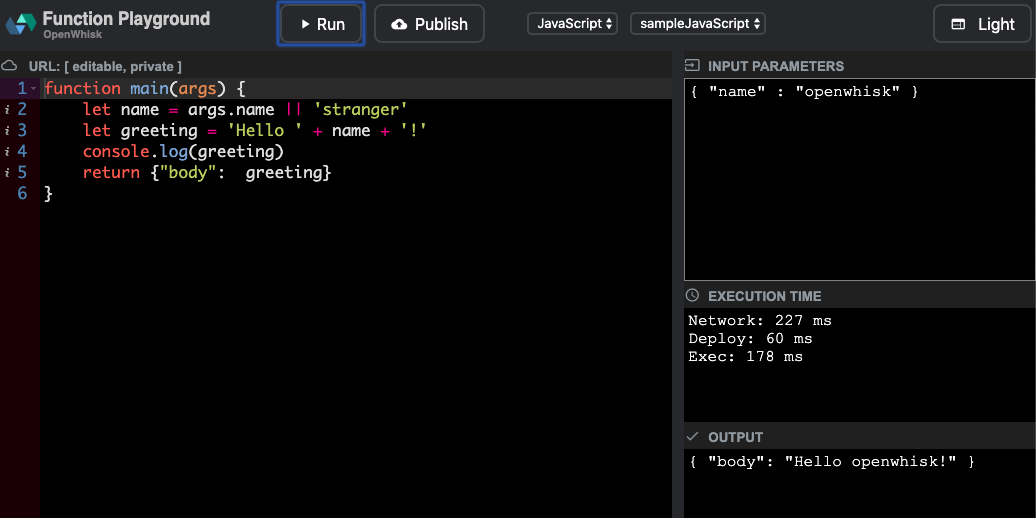
git clone https://github.com/apache/openwhisk.git

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. Playgrounds 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 [37]

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-има користећи манифест апликације.
* **OpenWhisk REST API** – након што се омогући OpenWhisk окружење, може се комуницирати са веб 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, једноставом 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].

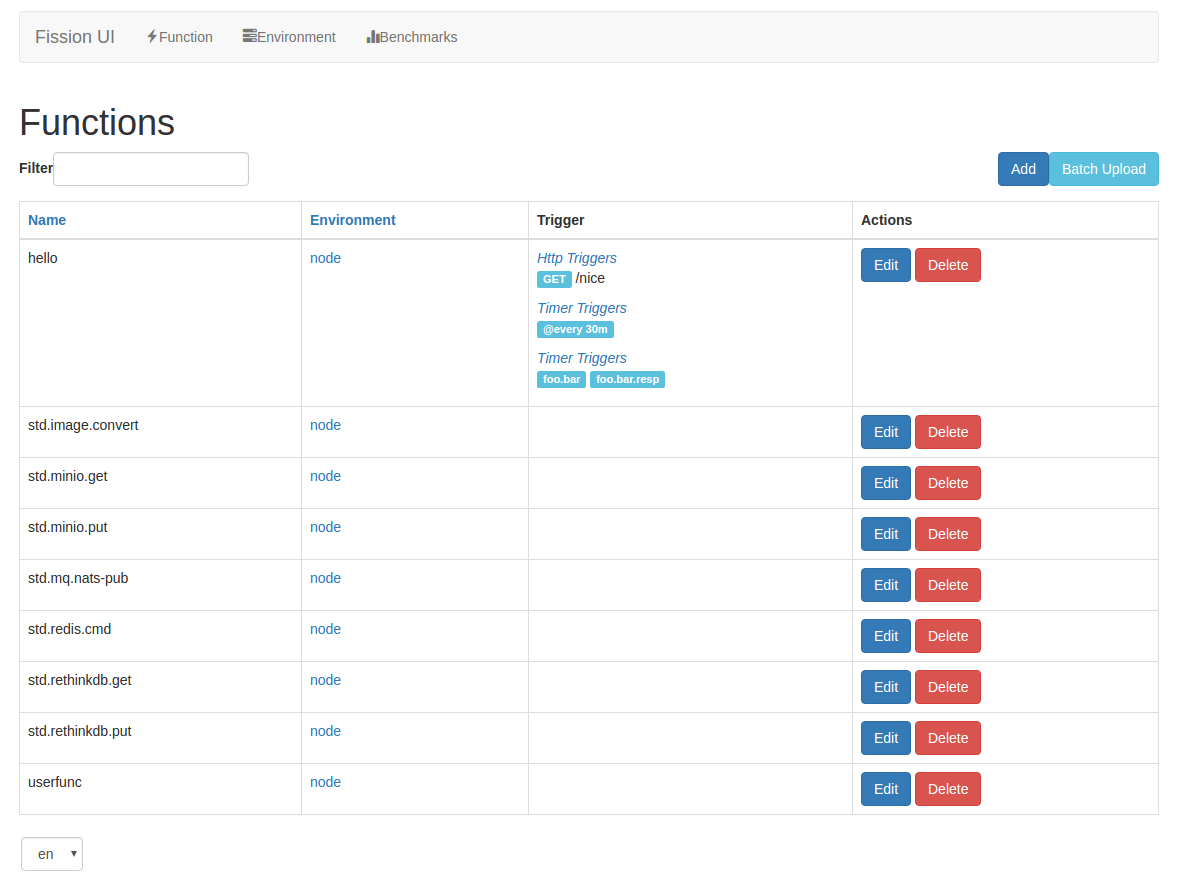
Платформа пружа многе бенефите, а неки од њих су [20]:

* **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 латенцију 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 *open source* платформа за повезивање, управљање и осигуравање микросервиса. Помоћу овог система корисници такође добијају могућност праћења употребе функција и праћења латенције 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 латенцију, 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.

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 [38]

#### 3.3.3.1. Инсталација

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.

Листинг 3.2. Инсталација употребом Helm пакета

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

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

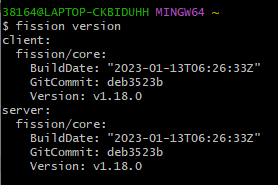
kubectl config set-context --current --namespace**=**$FISSION\_NAMESPACE

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

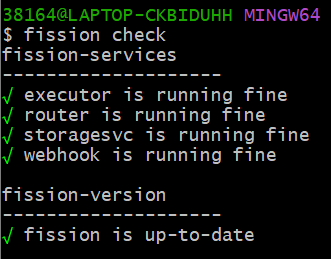
Након инсталације Fission платформе, остаје само инсталација Fission CLI алата који помаже при коришћењу платформе. Fission CLI се инсталира тако што се преузме бинарни фајл са званичног GitHub репозиторијума и премести у путању која је наведена у PATH системској променљивој.

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 3.5.



Слика 3.5. 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

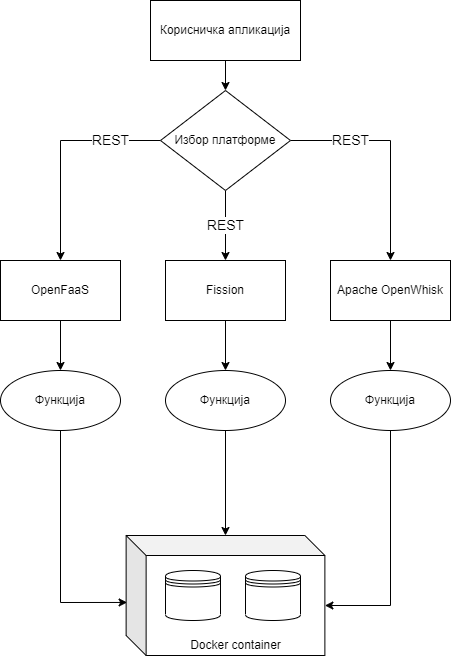
Апликација за пријаву гужве у саобраћају се састоји од две основне компоненте, а то су клијентска апликација и *serverless* платформе које имплементирају различите функционалности. Поред ове две основне компоненте, важно је споменути и базу података која се користи за складиштење података, као и протоколе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

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

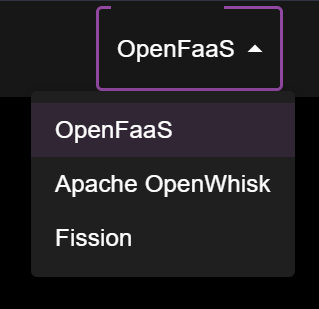
између корисничке апликације и *serverless* платформи коришћена је библиотека Axios [25]. Axios омогућава једноставно слање HTTP захтева, чиме је омогућена једноставна комуникација са платформама.

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. 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.

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

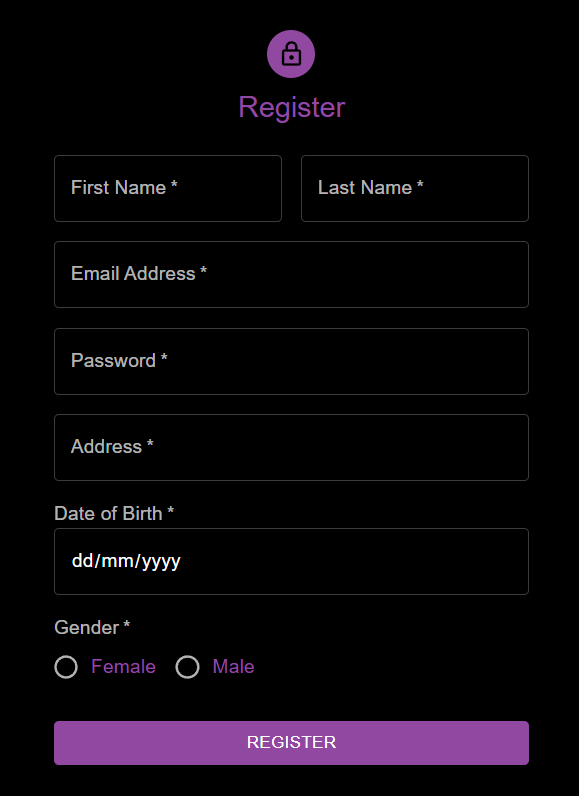
У овом делу биће размотрена и приказана имплементација система на све три платформе, кроз улоге које различити корисници имају у систему. Сви корисници у систему у свим моментима имају могућност промене *serverless* платформе са којом ће корисничка апликација комуницирати. Корисници могу одабрати платформу 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.

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

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'})

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'})

Листинг 5.2. Код за регистрацију корисника на Apache OpenWhisk платформи

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:

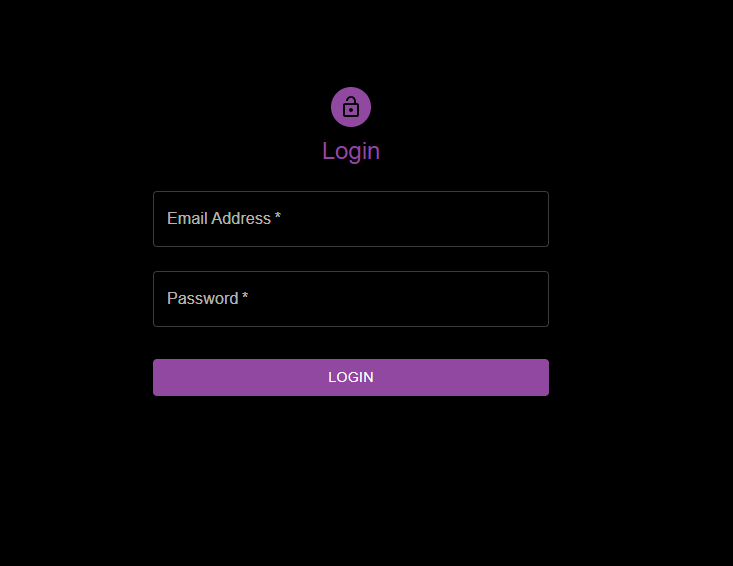
return {

'statusCode': 405,

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

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

}

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 email 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'})

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)

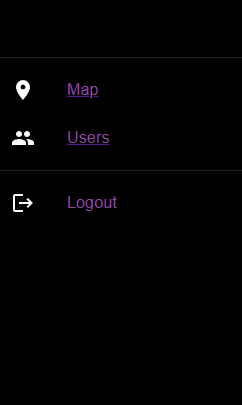
email = data["email"]

password = data["password"]

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

## 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“

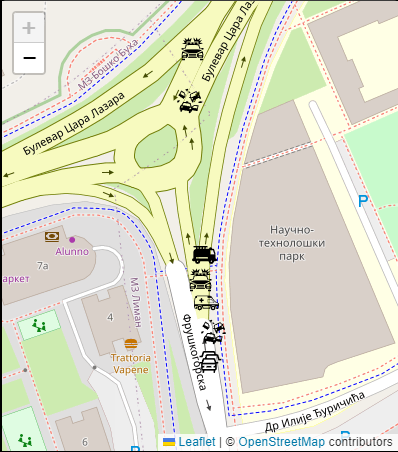


Слика 5.4. Приказ менија пријављеног корисника

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':

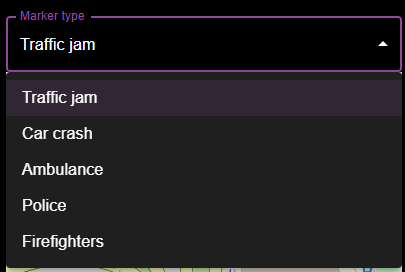
return get\_markers()

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

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

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':

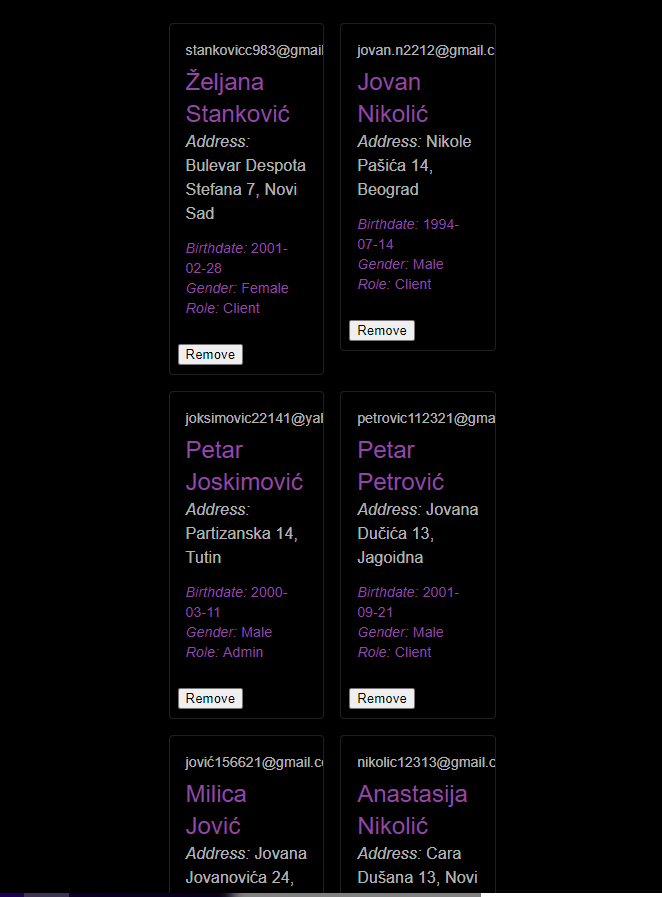
return add\_marker()

else:

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

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'})

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]

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

# 6. ЗАКЉУЧАК

У овом раду описана је и реализована апликација за пријаву гужве у саобраћају, са акцентом на имплементацију и упоређивање три open source платформе за serverless computing: 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 open source платформе за serverless computing, 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 завршном делу 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 даљем истраживању могло би да се фокусира на анализу перформанси при 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>

[2] What are some serverless use cases? | RedHat Topics

<https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless>

[3] Docker

<https://www.docker.com/>

[4] Developing with Docker | Docker

<https://www.docker.com/why-docker/>

[5] What container | Docker

<https://www.docker.com/resources/what-container/>

[6] Kubernetes

<https://kubernetes.io/>

[7] Kubernetes vs. Docker

<https://www.atlassian.com/microservices/microservices-architecture/kubernetes-vs-docker>

[8] Minikube

<https://minikube.sigs.k8s.io/docs/>

[9] OpenFaaS

<https://www.openfaas.com/>

[10] OpenFaaS | Docs

<https://docs.openfaas.com/>

[11] faas-cli

<https://github.com/openfaas/faas-cli>

[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/>

[19] Apache OpenWhisk | Docs

<https://openwhisk.apache.org/documentation.html>

[20] Fission

<https://fission.io/>

[21] Fission | Docs

<https://fission.io/docs/>

[22] React

<https://react.dev/>

[23] Material UI

<https://mui.com/material-ui/>

[24] React Leaflet

<https://react-leaflet.js.org/>

[25] Axios

<https://axios-http.com/>

[26] Python

<https://www.python.org/>

[27]Flask

<https://flask.palletsprojects.com/en/2.3.x/>

[28] PostgresSQL

<https://www.postgresql.org/>

[29] Psycopg2

<https://www.psycopg.org/>

[30] WSGI

<https://wsgi.readthedocs.io/en/latest/>

[31] A library that bridges the FaaS layer to the WSGI

<https://github.com/alexmilowski/flask-openwhisk>

[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>

[38] Fission UI

<https://github.com/fission/fission-ui>

[39] Javascript

<https://www.javascript.com/>

[40] Java

<https://www.java.com/en/>

[41] Go

<https://go.dev/>

[42] C#

<https://learn.microsoft.com/en-us/dotnet/csharp/>

[43] Swift

<https://www.swift.org/>

[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>.

[45] AWS Lambda

<https://aws.amazon.com/lambda/>

[46] Linux

<https://www.linux.org/>

[47] Windows

<https://www.microsoft.com/en-us/windows?r=1>

[48] Ubuntu

<https://ubuntu.com/>

[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>.

[50] containerd

<https://containerd.io/>

[51] CRIO

<https://kubernetes.io/docs/concepts/architecture/cri/>

[52] YAML

<https://yaml.org/>

[53] What is a Pod?

<https://kubernetes.io/docs/concepts/workloads/pods/>

[54] MacOS

<https://support.apple.com/macos>

[55] Oracle VM VirtualBox

<https://www.virtualbox.org/>

[56] VMware

<https://www.vmware.com/>

[57] Kernel Virtual Machine

<https://www.linux-kvm.org/page/Main_Page>

[58] REST

<https://restfulapi.net/>

[59] Prometheus

<https://prometheus.io/>

[60] Ruby

<https://www.ruby-lang.org/en/>

[61] Node.js

<https://nodejs.org/en>

[62] PHP

<https://www.php.net/>

[63] Express.js

<https://expressjs.com/>

[64] Eclipse Vert.x

<https://vertx.io/>

[65] ASP.NET Core

<https://learn.microsoft.com/en-us/aspnet/core/>

[66] FastAPI

<https://fastapi.tiangolo.com/>

[67] Django

<https://www.djangoproject.com/>

[68] Helm

<https://helm.sh/>

[69] Flux CLI

<https://fluxcd.io/flux/cmd/>

[70] Argo CD

<https://argo-cd.readthedocs.io/>

[71] Grafana

<https://grafana.com/>

[72] Apache Mesos

<https://mesos.apache.org/>

[73] Red Hat OpenShift

<https://www.redhat.com/en/technologies/cloud-computing/openshift>

[74] Scala

<https://www.scala-lang.org/>

[75] Deno

<https://deno.land/>

[76] Github

<https://github.com/>

[77] Fluentd

<https://www.fluentd.org/>

[78] Istio

<https://istio.io/>

[79] InfluxDB

<https://www.influxdata.com/>

# 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**: |  |
| Тип записа, **ТЗ**: |  |
| Врста рада, **ВР**: |  |
| Аутор, **АУ**: |  |
| Ментор, **МН**: |  |
| Наслов рада, **НР**: |  |
| Језик публикације,**ЈП**: |  |
| Језик извода, **ЈИ**: |  |
| Земља публиковања, **ЗП**: |  |
| Уже географско подручје, **УГП**: |  |
| Година, **ГО**: |  |
| Издавач, **ИЗ**: |  |
| Место и адреса, **МА**: |  |
| Физички опис рада, **ФО**: |  |
| Научна област, **НО**: |  |
| Научна дисциплина, **НД**: |  |
| Предметна одредница/кључне речи, **ПО**: |  |
| **УДК** |  |
| Чува се у, **ЧУ**: |  |
| Важна напомена, **ВН**: |  |
| Извод, **ИЗ**: |  |
| Датум прихватања теме, **ДП**: |  |
| Датум одбране, **ДО**: |  |
| Чланови комисије, **КО**: |  |
| председник |  |
| члан |  |
| ментор |  |
| Потпис ментора | |

# KEY WORDS DOCUMENTATION

|  |  |
| --- | --- |
| Accession number, **ANO**: |  |
| Identification number, **INO**: |  |
| Document type, **DT**: |  |
| Type of record, **TR**: |  |
| Contents code, **CC**: |  |
| Author, **AU**: |  |
| Mentor, **MN**: |  |
| Title, **TI**: |  |
| Language of text, **LT**: |  |
| Language of abstract, **LA**: |  |
| Country of publication, **CP**: |  |
| Locality of publication, **LP**: |  |
| Publication year, **PY**: |  |
| Publisher, **PB**: |  |
| Publication place, **PP**: |  |
| Physical description, **PD**: |  |
| Scientific field, **SF**: |  |
| Scientific discipline, **ND**: |  |
| Subject / Keywords, **S/KW**: |  |
| **UDC** |  |
| Holding data, **HD**: |  |
| Note, **N**: |  |
| Abstract, **AB**: |  |
| Accepted by sci. board on, **ASB**: |  |
| Defended on, **DE**: |  |
| Defense board, **DB**: |  |
| president |  |
| member |  |
| mentor |  |
| Mentor’s signature | |