

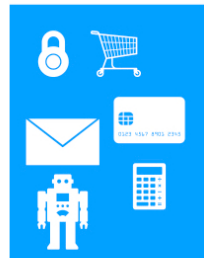
OpenFaaS

Function as a Service

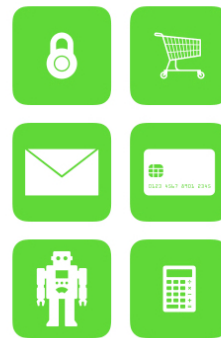
▶ Slavko Petrović 17345

Serverless arhitektura

- ▶ Serverless arhitektura je način razvoja i pokretanja aplikacija i servisa bez potrebe za održavanjem infrastrukture.
- ▶ Naziv serverless se koristi zbog toga što osoba odgovorna za sistem nema potrebe za naručivanjem, podešavanjem i održavanjem samog servera.
- ▶ Fokus je na aplikaciji, ne na infrastrukturi.



Monolith



Microservices



FaaS

Function as a Service - FaaS

- ▶ FaaS je podskup serverless računarstva koji je fokusiran na event-driven triggerima gde se funkcija, kod, pokreće kao odgovor na događaje ili zahteve.
- ▶ Najčešće se koristi kod izgradnji mikroservisnih aplikacija.
- ▶ Server koji izvršava funkcije je aktivan samo tokom njenog izvršenja.
- ▶ Kada se funkcija izvrši, server se gasi i omogućavajući da se računarski resursi dodele negde drugde.
- ▶ Plaćaju se samo resursi koje koristimo, kada ih koristimo.

Open source FaaS platforme

- ▶ Najpoznatije open source FaaS platforme su:

- ▶ OpenFaaS
- ▶ Kubeless
- ▶ OpenWhisk
- ▶ Knative
- ▶ Fission
- ▶ Fn
- ▶ Nuclio

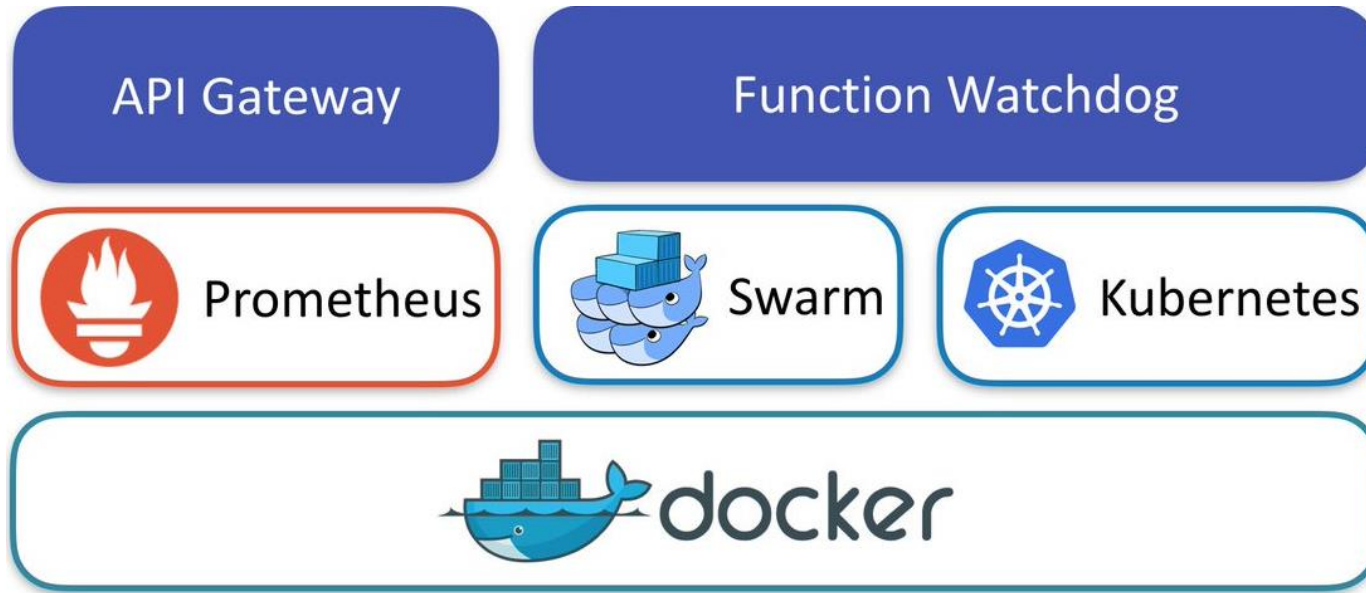


OpenFaaS

- ▶ OpenFaaS je open source alat koji omogućava programerima da lako razviju event-driven funkcije i mikroservise na Kubernetes-u bez ponavljanja koda.
- ▶ Podržava većinu programskih jezika, Go, Java, Python, C# / .Net, Ruby...
- ▶ Osnovao ga je Alex Ellis 2016. godine.
- ▶ Pisan u Golang jeziku .
- ▶ MIT licenciran.



Cloud Native Stack

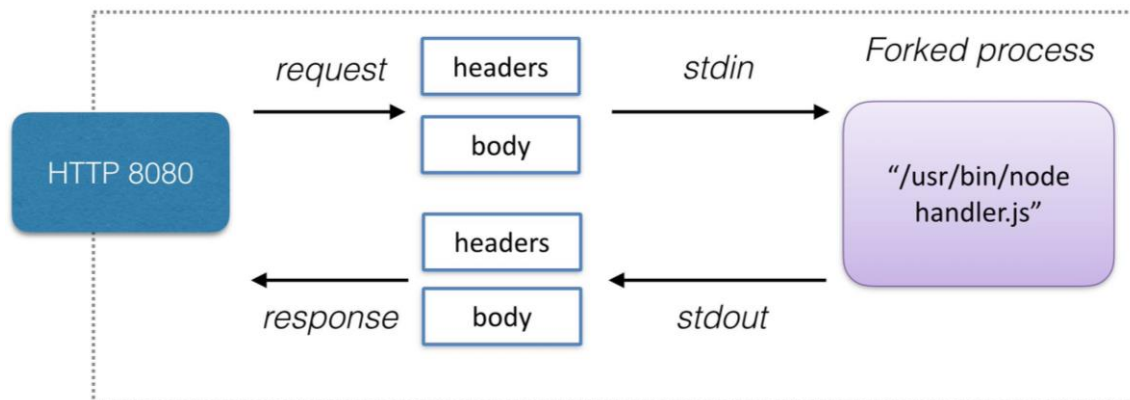


- ▶ PLONK stack
 - ▶ Prometheus
 - ▶ Linux/Linkerd
 - ▶ OpenFaaS
 - ▶ NATS
 - ▶ Kubereneets

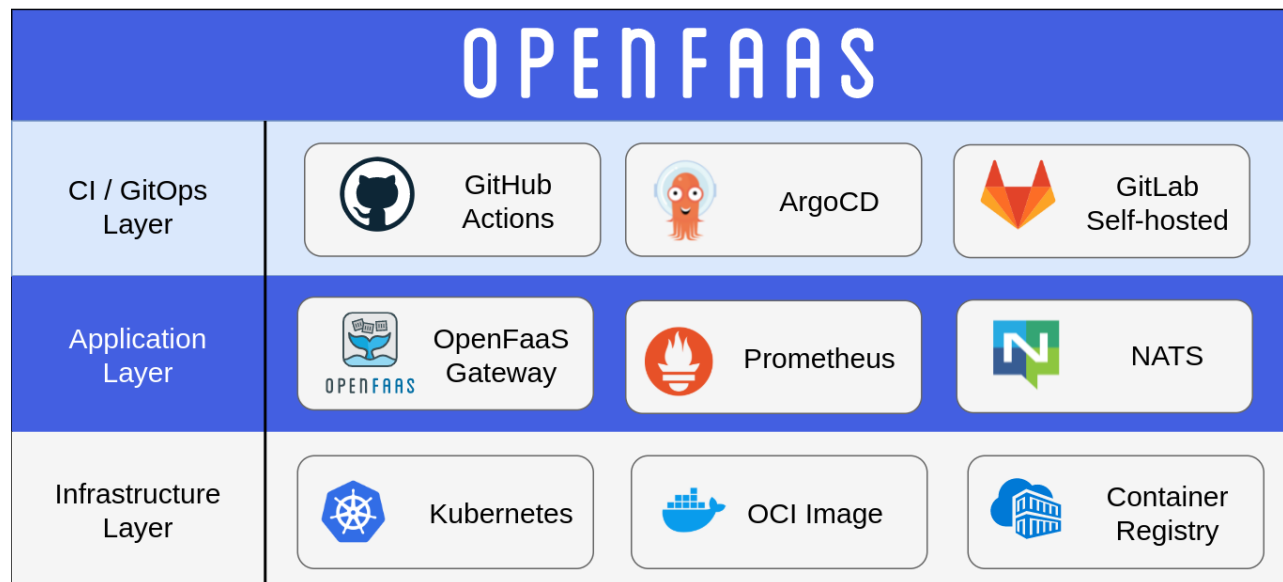
OpenFaaS watchdog

- ▶ Function watchdog je ugrađen u svaki kontejner i to omogućava da bilo koji kontejner postane serverless(mali Golang HTTP server).
- ▶ OpenFaas watchdog je odgovoran za pokretanje i nadgledanje funkcija u OpenFaas-u. Bilo koja binarna datoteka može postati funkcija korišćenjem watchdog-a.

Watchdog

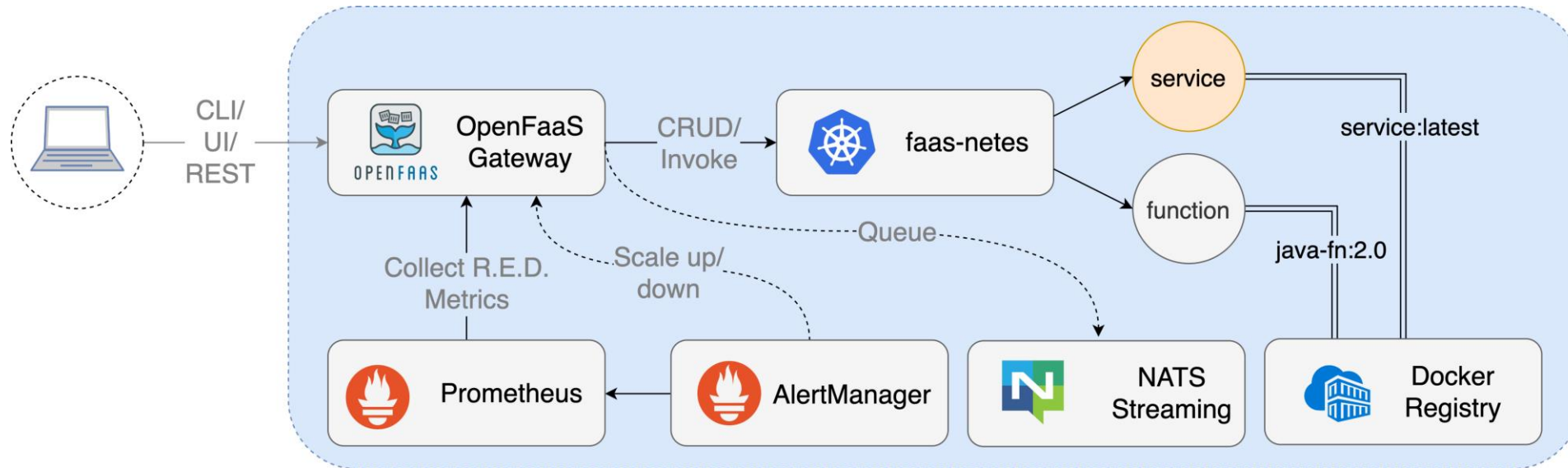


OpenFaaS stack



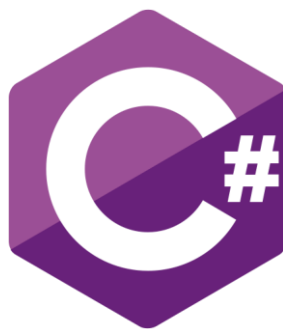
- ▶ Application layer:
 - ▶ OpenFaas Gateway - pruža REST API za upravljanje funkcijama, snimanje metrika i skaliranje funkcija.
 - ▶ Prometheus - pruža metrike i omogućava auto-skaliranje.
 - ▶ NATS - koristi se za asinhrono izvršenje funkcija.

Conceptual workflow

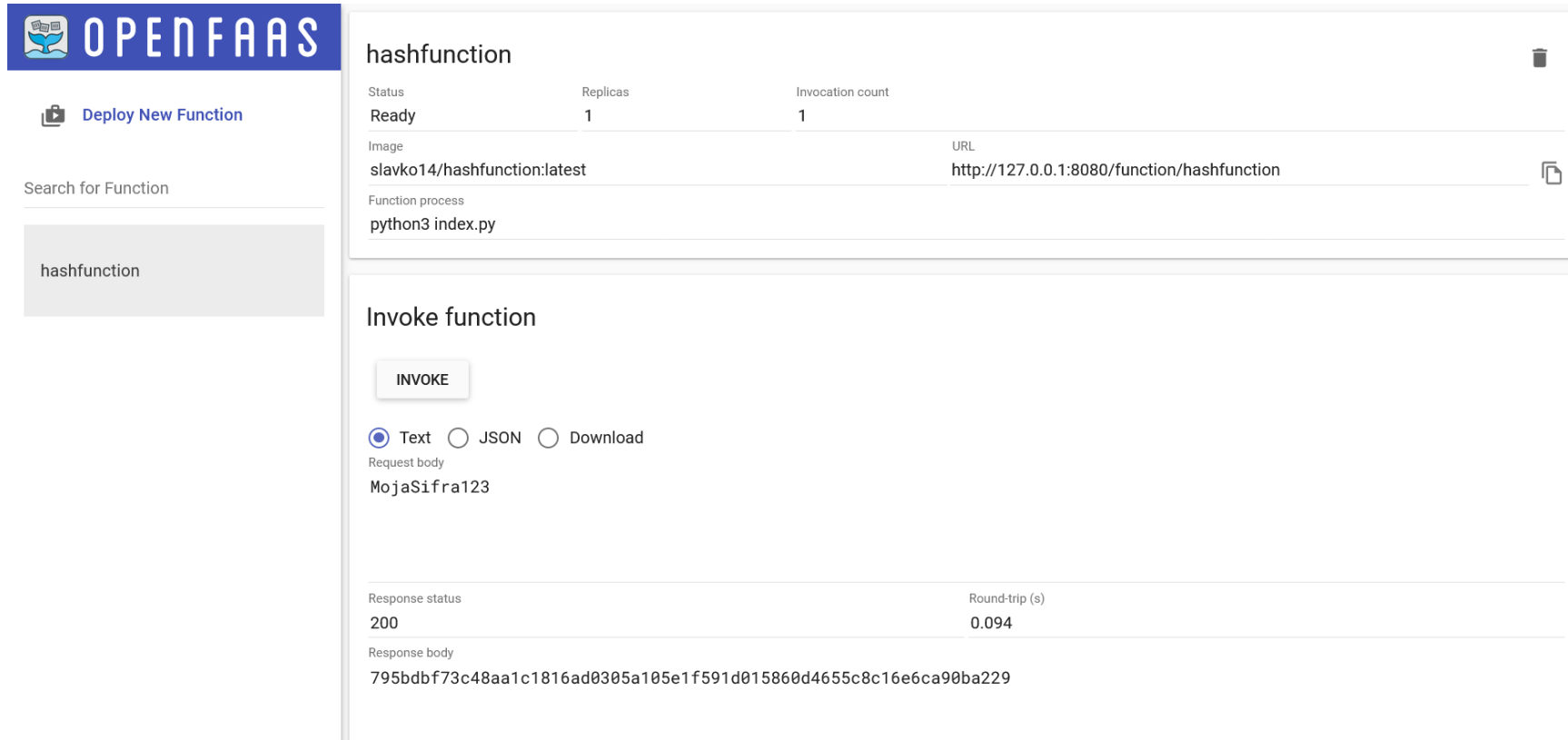


OpenFaas CLI

- ▶ OpenFaas ima moćan CLI koji se može koristiti za build-ovanje i deployment funkcija na OpenFaas.
- ▶ Mi možemo da napravimo OpenFaaS funkcije iz skupa podržanih jezičkih šablona(Node, Python, CSharp, Ruby, ...).
- ▶ To znači da pišemo samo funkciju u jednom fajlu(handler.py, handler.js, ...), a CLI radi ostalo da kreira Docker sliku.



OpenFaas UI



The screenshot displays the OpenFaas UI interface. On the left sidebar, there is a logo and the text 'OPENFAAS', a 'Deploy New Function' button, and a search bar containing 'hashfunction'. The main panel shows details for the 'hashfunction' service, including its status (Ready), replicas (1), invocation count (1), image (slavko14/hashfunction:latest), URL (http://127.0.0.1:8080/function/hashfunction), and function process (python3 index.py). Below this, the 'Invoke function' section features an 'INVOKE' button and radio buttons for 'Text' (selected), 'JSON', and 'Download'. The request body is 'MojaSifra123'. The response details show a status of 200, a round-trip time of 0.094s, and a response body consisting of a long alphanumeric string.

Status	Replicas	Invocation count
Ready	1	1

Image	URL
slavko14/hashfunction:latest	http://127.0.0.1:8080/function/hashfunction

Function process
python3 index.py

Invoke function

☒ Text ☐ JSON ☐ Download

Request body
MojaSifra123

Response status	Round-trip (s)
200	0.094

Response body
795bdbf73c48aa1c1816ad0305a105e1f591d015860d4655c8c16e6ca90ba229

- ▶ Preko OpenFaaS UI možemo ručno deploy-ovati funkcije kao i testirati funkcije koje su već deploy-ovane.

OpenFaas prednosti i mane

- ▶ Pojednostavljuje izgradnju sistema
- ▶ Lakše otklanjanje grešaka
- ▶ Lakše dodavanje novih funkcionalnosti
- ▶ Omogućava da kod pokrenemo bilo kada, bilo gde, bilo kojim programskim jezikom
- ▶ Skalabilnost
- ▶ Community
- ▶ Dokumentacija

OpenFaas prednosti i mane

- ▶ Dugo vreme hladnog starta za neke programske jezike
- ▶ Vreme pokretanja kontejnera zavisi od provajdera
- ▶ Kasnjenje
- ▶ Limitirani životni vek funkcije

OpenFaaS jednostavan primer

- ▶ Da bi krenuli sa pisanjem i testiranjem funkcija na Windows operativnom sistemu potreban nam je Docker i GitBash preko koga ćemo skinuti ostale alate.
- ▶ Pomocu GitBash-a skidamo arkade koji nam omogućava da skinemo ostale alate. [curl -SLsf https://get.arkade.dev/ | sh](https://get.arkade.dev/)
- ▶ Potom redom skidamo redom Kubernetes, kind i OpenFaaS CLI
- ▶ [arkade get kubectl, arkade get kind, arkade get faas-cli](#)
- ▶ Kreiramo kind kluster - [./kind create cluster](#)
- ▶ Na klasteru instaliramo OpenFaaS - [arkade install openfaas](#)
- ▶ Pokrećemo cluster na portu 8080
- ▶ [./kubectl port-forward -n openfaas svc/gateway 8080:8080](#)

OpenFaaS jednostavan primer

- ▶ Potom generišemo šifru i prikazujemo je:
- ▶ `PASSWORD=$(kubectl get secret -n openfaas basic-auth -o jsonpath="{.data.basic-auth-password}" | base64 --decode; echo)`
- ▶ `Echo -n $PASSWORD`
- ▶ Zatim se povezujemo na '<http://127.0.0.1:8080/ui/>' kao username koristimo admin i gore generisanu šifru
- ▶ Sada možemo napisati i deploy-ovati funkciju komandama. Prvo generišemo šablon funkcije u izabranom programskom jeziku
- ▶ `./faas-cli new "name" --lang "language"`

OpenFaaS jednostavan primer

- ▶ U folderu kako smo nazvali našu aplikaciju menjamo fajl handler.xy, pišući našu funkciju.

```
import hashlib

def handle(req):
    hash_object = hashlib.sha256(str(req).encode('utf-8'))
    hex_dig = hash_object.hexdigest()
    return hex_dig
```

- ▶ U name.yml fajlu treba dodati DockerID/ ispred naziva docker slike.

```
version: 1.0
provider:
  name: openfaas
  gateway: http://127.0.0.1:8080
functions:
  hashfunction:
    lang: python3
    handler: ./hashfunction
    image: slavkol4/hashfunction:latest
```


OpenFaaS jednostavan primer

- ▶ Nakon što smo to izmenili trebalo bi da se prijavimo i preko OpenFaaS CLI i potom odraditi deployment.
- ▶ [./faas-cli login -password "šifra"](#)
- ▶ [./faas-cli up -f "naziv.yml"](#)
- ▶ Sada možemo testirati naše funkcije preko OpenFaaS UI

The screenshot displays the OpenFaaS web interface. On the left, there's a sidebar with the 'OPENFAAS' logo and a 'Deploy New Function' button. Below it is a search bar with 'hashfunction' entered. The main panel shows details for the 'hashfunction' function, including its status (Ready), replicas (1), invocation count (1), image (slavko14/hashfunction:latest), and URL (http://127.0.0.1:8080/function/hashfunction). Below this, there's an 'Invoke function' section with an 'INVOKE' button and radio buttons for 'Text' (selected), 'JSON', and 'Download'. The request body is 'MojaSifra123'. At the bottom, the response status is 200, the round-trip time is 0.094s, and the response body is a long alphanumeric string.

Status	Replicas	Invocation count
Ready	1	1

Image: slavko14/hashfunction:latest
URL: http://127.0.0.1:8080/function/hashfunction
Function process: python3 index.py

Invoke function

☒ Text ☐ JSON ☐ Download

Request body
MojaSifra123

Response status	Round-trip (s)
200	0.094

Response body
795bdbf73c48aa1c1816ad0305a105e1f591d015860d4655c8c16e6ca90ba229

Literatura

- ▶ <https://docs.openfaas.com/>
- ▶ <https://yankee.dev/serverless-function-openfaas-kubernetes-locally>
- ▶ <https://github.com/openfaas/faas>
- ▶ <https://www.cncf.io/blog/2020/04/13/serverless-open-source-frameworks-openfaas-knative-more>

Hvala na paznji