

UNIVERZITET U NIŠU  
ELEKTRONSKI FAKULTET

**KNATIVE – FUNCTION AS A SERVICE PLATFORMA**

Seminarski rad

Predmet: Servisno-orijentisane arhitekture

Student: Mentor:

Jovana Spasić, br. ind. 17407 Prof. dr Dragan Stojanović

Niš, maj 2022. godina

**SADRŽAJ**

[UVOD 4](#_Toc103874490)

[Knative Eventing 5](#_Toc103874491)

[**Komponente** 5](#_Toc103874492)

[***Usage Patterns*** 5](#_Toc103874493)

[Knative Serving 6](#_Toc103874494)

[***Deployment* model** 6](#_Toc103874495)

# **UVOD**

FaaS (*Function as a Service*) predstavlja vrstu usluge računarstva u oblaku koja omogućava programerima da kreiraju, pokreću i upravljaju paketima aplikacija kao funkcijama bez potrebe da održavaju sopstvenu infrastrukturu. FaaS je tzv. *event-driven* model izvršavanja koji radi u *stateless* kontejnerima i kod koga funkcije upravljaju *server-side* logikom i stanjem korišćenjem servisa, tj. usluga ponuđenih od strane FaaS provajdera.

Hosting aplikacije na Internetu uobičajeno zahtevaju obezbeđivanje i upravljanje virtuelnim ili fizičkim serverom, kao i upravljanje operativnim sistemom i hosting procesima *web* servera. Kada se koristi FaaS, tada fizičkim hardverom, operativnim sistemom virtuelne mašine i *web* serverom upravlja *cloud* servis provajder. Time je omogućeno programeru da se fokusira isključivo na pojedinačne funkcije u kodu.

FaaS rešenja je moguće pronaći na svim velikim javnim *cloud* platformama. Neki popularniji primeri uključuju:

* *IBM Cloud Functions*
* *Amazon’s AWS Lambda*
* *Google Cloud Functions*
* *Microsoft Azure Functions*
* *OpenFaaS*

Postoji nekoliko dobrih praksi koje programeri mogu slediti kako bi lakše i efikasnije koristili FaaS:

* Svaka funkcija obavlja samo jednu akciju – FaaS funkcije je potrebno dizajnirati tako da rade samo određeni deo posla kao odgovor na događaj. Pisanje efikasnog, *lightweight* koda sa limitiranim *scope*-om omogućava brže učitavanje i izvršavanje funkcija.
* Funkcije se ne trebaju međusobno pozivati – Previše funkcija će povećati troškove i ukloniti vrednost izolacije funkcija.
* Koristiti što je manje moguće biblioteka u funkcijama – Korišćenje prevelikog broja biblioteka može usporiti funkcije i otežati njihovo skaliranje.

Kubernetes i Knative predstavljaju implementaciju „vodovoda“ iza FaaS-a. Kubernetes je *open-source* alat za orkestraciju kontejnera i predstavlja ključnu aplikaciju za upravljanje *cloud* aplikacijama. Knative je automatizovani sistem koji pomaže programerskim timovima pri upravljanju i održavanju procesa unutar Kubernetes-a. Njegova svrha jeste da pojednostavi, automatizuje i nadgleda *deployment* Kubernetes-a, tako da timovi provode manje vremena u održavanju, a više u samom razvoju aplikacije. Knative preuzima zadatke koji se ponavljaju i koji su vremenski zahtevni uklanjajući „prepreke“ i kašnjenja.

Knative sadrži dve glavne komponente koje pomažu timovima koji rade sa Kubernetes-om, a to su Knative Eventing i Knative Serving. Knative Eventing omogućava programerima da podese detaljne akcije koje će biti *trigger*-ovane specifičnim događajima u širem okruženju. Knative Serving automatski upravlja kreiranjem i skaliranjem usluga kroz Kubernetes, uključujući skaliranje na nulu. Ove komponente za cilj imaju oslobađanje resursa koje bi timovi potrošili na upravljanje sistemima.

# **Knative Eventing**

Knative Eventing obezbeđuje alate za rutiranje događaja od njihovog proizvođača pa do prijemnika, omogućavajući time programerima da koriste *event-driven* arhitekture u svojim aplikacijama.

Resursi su slabo spregnuti i mogu se nezavisno razvijati i koristiti. Svaki proizvođač može generisati događaje pre nego što se „jave“ aktivni potrošači događaja koji osluškuju. Takođe, svaki potrošač može pokazati interesovanje za događajem ili klasom događaja, čak i pre nego što proizvođači kreiraju te događaje.

Koriste se standardni HTTP POST zahtevi za slanje i primanje događaja između proizvođača događaja i prijemnika. Ovi događaji su u skladu sa CloudEvents specifikacijama, što omogućava kreiranje, parsiranje i primanje događaja u bilo kom programskom jeziku.

## **Komponente**

*Event-driven* arhitektura zasniva se na konceptu odvajanja veza između proizvođača događaja i potrošača događaja. Koristi se HTTP protokol. Potrošač takođe može biti konfigurisan da odgovara na HTTP zahteve slanjem događaja koji odgovaraju odgovoru.

*Event sources* predstavljaju primarne proizvođače događaja. Događaji se šalju potrošaču.

*Broker and Trigger* obezbeđuju „*event mesh*“ model, koji dozvoljava proizvođaču događaja da isporuči događaje *broker*-u, koji ih onda distribuira uniformno potrošačima korišćenjem *trigger*-a. Benefiti ovakvog pristupa su sledeći:

* Potrošači se mogu registrovati za specifične tipove događaja bez potrebe za pregovaranjem direktno sa proizvođačima događaja.
* *Event routing* može biti optimizovano od strane platforme koristeći specificirane filtere uslova.

*Channel and Subscription* obezbeđuju „*event pipe*“ model koji transformiše i rutira događaje između kanala korišćenjem pretplata. Prethodno pomenuti model je prikladan za *event pipeline* gde događaji sa jednog sistema trebaju biti transformisani i rutirani do sledećeg procesa.

*Event Registry* je sačinjen od kolekcije tipova događaja. Tipovi događaji koji se nalaze unutar registra sadrže neophodne informacije za potrošača kako bi se kreirao *trigger* bez potrebe za pribegavanjem nekom drugom mehanizmu.

## ***Usage Patterns***

*Source to Service* pruža najjednostavnije korišćenje Knative Eventing-a. Obezbeđuje jedan Sink, tj. servis za primanje događaja, bez čekanja, povratnog pritiska i filtriranja. Ovakav vid komunikacije ne podržava odgovore, što znači da se odgovor od Sink servisa ignoriše. Odgovornost Event Source-a je samo da dostavi poruku bez čekanja na odgovor od Sink-a.

*Channel and Subscription* definiše kanal koji može biti povezan sa različitim stvarima u pozadini kao što su In-Memory, Kafka, GCP PubSub za izvor događaja. Svaki kanal može imati više pretplatnika u formi Sink servisa, koji mogu primati poruke događaja i po potrebi ih obrađivati. Svaka poruka sa kanala sa formatira kao CloudEvent i šalje dalje kroz lanac ostalim pretplatnicima na procesiranje. Ovaj obrazac korišćenja nema mogućnost filtriranja poruka.

*Broker and Trigger* sličan je prethodnom obrascu, s tim što podržava filtriranje događaja. Reč je o metodu koji dozvoljava pretplatnicima da pokažu interesovanje za određeni skup poruka koje ulaze u *broker*. Za svaki *broker*, Knative Eventing implicitno kreira kanal. *Trigger* se samostalno pretplaćuje na *broker* i primenjuje filter na poruke *broker*-a. Filteri se primenjuju na CloudEvent atribute poruke, pre nego što se dostave zainteresovanim servisima.

# **Knative Serving**

Idealan je za pokretanje servisa unutar Kubernetes-a, pošto obezbeđuje pojednostavljenu sintaksu za *deployment*, sa automatskim skaliranjem na nulu i skaliranjem na osnovu HTTP opterećenja. Knative platforma će upravljati *deployment*-om, revizijama, umrežavanjem i skaliranjem.

„Izlaže“ servis preko HTTP URL-a i ima mnogo sigurnih podešavanja u konfiguraciji. Za mnoge upotrebe u praksi je možda potrebno izvršiti promenu postojećih podešavanja, tako da odgovaraju potrebama.

## ***Deployment* model**

Tokom *deployment*-a KS servisa, KS kontroler kreira konfiguraciju, reviziju i rutu. Ruta predstavlja URL preko koga je moguće pristupiti servisu. Konfiguracija održava željeno stanje, omogućava razdvajanje koda i konfiguraciju koristeći principe razvoja aplikacija sa dvanaest faktora. Na osnovu željenog stanja, kontroler konfiguracije kreira novi Kubernetes *deployment* za aplikaciju. Svaka promena u konfiguraciji rezultuje novim *deployment*-om. Zbog prethodno pomenutog principa, svaka promena konfiguracije aplikacije kreira novu reviziju. Revizija je slična *version control* oznaci ili labeli i nepromenljiva je. Svaka revizija je povezana sa odgovarajućim Kubernetes *deployment*-om, što omogućava da se aplikacija vrati na prethodnu verziju koja je bila validna.