Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/05BE05Milanovic

# INSTANA ALAT ZA NADGLEDANJE PERFORMANSI APLIKACIJA I JAVA ATTACH API

# INSTANA TOOL FOR MONITORING APPLICATION PERFORMANCE AND JAVA ATTACH API

Igor Milanović, Milan Vidaković, *Fakultet tehničkih nauka, Novi Sad*

**Oblast – ELEKTROTEHNIKA I RAČUNARSTVO**

**Kratak sadržaj:** *U ovom radu je analiziran način funkcionisanja Instana alata za nadgledanje aplikacija i infrastrukture sa akcentom na Java attach API. Specifi-cirani su zahtevi i dizajn, implementirana je aplikacija koja prikuplja i prikazuje parametre korisničkih aplikacija.*

***Abstract***: *This paper analyses the Instana tool for monitoring of applications and infrastructure with more details about Java attach API. It specifies the requirements and the design and describes the implementation of application which gathers client's application parameters and displays them.*

**Ključne reči:** *Instana, Java, agent, microservices, performance.*

## 1. UVOD

U oblasti informacionih tehnologija i upravljanja sistemima, upravljanje performansama aplikacija (APM - *Application performance management*) predstavlja praćenje i upravljanje performansama i dostupnosti softverskih aplikacija. APM nastoji otkriti, dijagnostifikovati i predvideti složene probleme u performansama aplikacija kako bi održao očekivani nivo usluge. Naglim povećanjem potrebe za skalabilnošcu sistema, pojavom i velikom popularnošću računarstva u oblaku (cloud computing) i distribuiranih servisa javlja se više nego ikada potreba za merenjem i nadgledanjem procesa unutar sistema.

U današnje vreme uopšte nije retkost da se aplikacija sastoji od dvocifrenog broja servisa koji međusobno ostvaruju interakciju i na taj način ispunjavaju zahteve korisnika. Jedna od najvećih kompanija koja koristi mikroservisnu arhitekturu je svakako Netflix, koja trenutno koristi oko 700 različith vrsta mikroservisa [1].

Kada se ta brojka pomnoži sa koeficijentom skalabilnosti, lako se dolazi do cifre od nekoliko hiljada zasebnih servisa koji mogu da se nalaze u različitim ekosistemima. Glavni cilj APM sistema je povećanje kvaliteta iskustva krajnjeg korisnika (user experience) aplikacije koja se prati. Translacija prikupljenih metrika u vrednost od koje korisnik ima benefita je najvažniji zadatak svakog APM sistema [2].

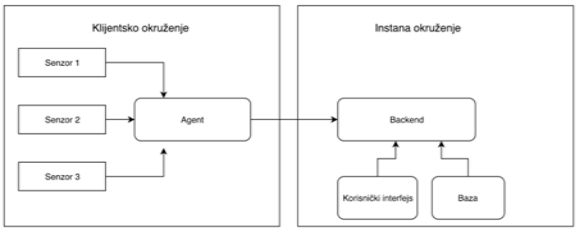
**NAPOMENA:**
**Ovaj rad je proistakao iz master rada čiji mentor je bio prof. dr Milan Vidaković.**

## 2. PREGLED SISTEMA

Osnovni sistem se sastoji od: Senzora, Agenta, *Backenda* i komponente za vizuelizaciju u okviru pretraživača koju ćemo nazvati UI (*User interface*, korisnički interfejs).

Senzori su komponente koje se izvršavaju u sklopu klijentske virtualne mašine tako što se infiltriraju u bajt kod u fazi izvršavanja aplikacije i prikupljaju samo promene metrika klijentskog koda koji presleđuje dalje agentu.Agent je komponenta koja je vezana za određenu virtualnu mašinu, odnosno programski jezik. Zbog jednostavnosti u daljem tekstu će biti opisivan agent za Java virtualnu mašinu. Odgovornost agenta je da detektuje da li na klijentskoj virtualnoj mašini postoji tehnologija za koju postoji odgovarajući senzor, i ako je uslov ispunjen, da se senzor instancira. Nakon što je proverio osnovni uslov, agent mora da prati životni vek senzora ne dozvoljavajući mu da prestane sa radom. Pored ove odgovornosti, agent mora biti u stalnoj vezi sa backendom da bi primenjivao razne promene u podeša-vanjima i da bi dobavljao nove verzije izvršnih kodova senzora. Sve metrike koje stignu sa agenta na backend se moraju obraditi. Pod pojmom obrada podrazumeva se da li su klijentska pravila ispunjena, te stoga da li je potrebno da se aktivira određeni alarm. *Backend* pored pomenute funkcionalnosti takođe čuva metrike u bazi i time omogućava klijentu da ih pregleda.

Važno je napomenuti da klijent ima mogućnost da dodaje proizvoljna pravila koja su vezana za određenu tehnologiju. Na primer, pravilo može biti: ako je popunjenost memorije prešla devedeset procenata da se obavesti odgovorno lice. Određena pravila dolaze sa sistemom i njih korisnik ne može menjati; na taj način klijent dobija trenutnu vrednost od Instane bez potrebe da kreira svoja pravila. Na slici 1. je prikazana organizacija




Slika 1 – *Prikaz organizacije kompon*enti

komponenti koje su objašnjene u tekstu iznad. Komponente su prikazane unutar okruženja u kome se izvršavaju. Važno je napomenuti da se sve komponente mogu izvršavati i samo u klijentskom okruženju ukoliko klijent želi takav pristup.

## 3. KORIŠĆENE TEHNOLOGIJE OSGI i Karaf

OSGi specifikacija opisuje modularni sistem i servisnu platformu za Java programski jezik koji implementira kompletan i dinamičan komponentni model što ne postoji u Java VM okruženjima. Aplikacije ili komponente, koje dolaze u obliku paketa, mogu se daljinski instalirati, pokrenuti, zaustaviti, ažurirati i deinstalirati bez potrebe za ponovnim pokretanjem. Upravljanje životnim ciklusom aplikacija se implementira putem API-ja koji omogućavaju daljinsko preuzimanje i upravljanje. Agent se pokreće pomoću Karafa.

Karaf je okruženje za izvršavanje aplikacija. Svaki senzor je opisan kao komponenta u Karafu. Karaf u toku izvršavanja odlučuje da li je potrebno da se aktivira određeni senzor u zavisnosti da li su ispunjeni unapred specificirani uslovi za pokretanjem senzora za određenu tehnologiju. Kada se uslov ispuni, Karaf preuzima paket za senzor sa dostupnog servera i pokušava njegovo pokretanje bez ometanja drugih komponenti. U praksi to znači da je interferencija između različitih senzora svedena na minimum. Kao što prati da li ima potrebe da se pokrene određeni senzor tako isto prati da li je potrebno ugasiti određeni senzor. Pored toga što se bavi životnim vekom senzora, Karaf obezbeđuje i sve zavisnosti koje su neophodne senzoru za pravilno fukncionisanje. Misli se pre svega na zahtevane biblioteke. Karaf takođe omogućava i verzionisanje senzora, što omogućava lako objavljivanje novih verzija senzora i takođe omogućava postojanje specifičnih senzora koji su zahtevani od određenih klijenata i vidljivi su samo njihovim okruženjima

**Bajt kod instrumentacija**
Počevši od Jave 5, JDK omogućava instrumentaciju bajt koda (bytecode). Java bajt kod je rezultat kompilacije Java programa, posrednog predstavljanja programa koji je nezavisan o hardveru na kome se izvršava. Ova tehnika ima za cilj mogućnost modifikacije bajt koda koji se učitava u JVM i izvršava na njemu - na primer, njegovo proširenje sa dodatnim instrukcijama ili druge promene izvornog bajt koda. Treba napomenuti da instrumentacija bajt koda ne uzrokuje promenu izvornih resursa, .class fajlova. On manipuliše bajtkodom on-the-fly (u letu, u tok izvršavanja koda) u vreme kada class loader pokušava da pristupi i učita bajt kod odgovarajuće klase u JVM, proširujući ili zamenjujući bajt kod dobijen od originalnog resursa, sa svojom instrumentisanom verzijom.
Instrumentacijski interfejs pruža mogućnosti za dodavanje prilagođene implementacije transformatora koja će se aktivirati kada se bajt kod klase učita u JVM, i može proširiti ili zameniti izvorni bajt kod klase pomoću prilagođenog.

**Java agent**
S obzirom da u realnim uslovima nemamo često mogućnost da restartujemo aplikaciju koju želimo da nadgledamo da bi se klase ponovno učitale i time omogućile da infiltriramo kod za instrumentaciju potrebno nam je drugačije rešenje. Koncept koji podrazumeva odvajanje koda za instrumentaciju i same aplikacije postoji u JVM već neko vreme i poznat je kao Java agent. Java agent je aplikacija se obično isporučuje kao samostalna JAR datoteka (opciono sa dodatnim potrebnim zavisnostima), koja sadrži implementaciju instrumentacijske logike i može se povezati sa Java aplikacijama radi njihove instrumentacije. Postoje dva načina kako se Java agent može pokrenuti i učitati u instrumentovani JVM. To su staticki i dinamicki:

- Statičko pokretanje agenta tokom JVM pokretanja se postiže korišćenjem dodatnog JVM argumenta "-javaagent" i određivanjem lokacije JAR datoteke agenta kao vrednosti ovog argumenta (opciono, ako agent prihvati bilo koje parametre ili opcije, oni se takođe mogu proslediti kao deo argumenta): -javaagent: jarpath. Pomoću ovog pristupa može se učitati više agenata - potrebno je navesti nekoliko unosa argumenta "-javaagent", pri čemu se svaki od njih odnosi na pojedinačno učitanog agenta. Ako se to uradi, agenti će biti učitani u sekvenci dok se odgovarajući argumenti "-javaagent" pojavljuju u JVM listi argumenata.

- Dinamičko pokretanje podrazumeva pokretanje agenta nakon pokretanja JVM i njegovo povezivanje sa nadgledanom aplikacijom. Ovo se postiže korišćenjem API Attach, koji je jedan od dijagnostičkih interfejsa koji su dostupni u modernim JVM. Ideja ovog pristupa je da pomoću API možemo da se povežemo sa JVM (priključimo se na određeni port, slično kao debug način) i učitamo agent iz određene JAR datoteke sa neophodnim neobaveznim argumentima, bukvalno u bilo koje vreme izvršavanja Java aplikacije

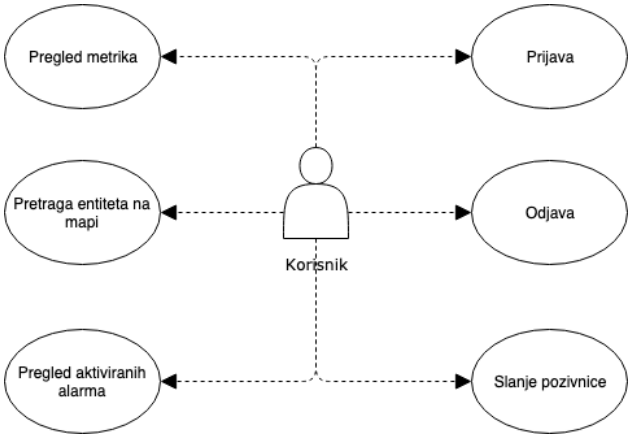
**Byte buddy**
Java instrumentacija kakvu koristi Instana ne bi bila moguća bez alata poput Byte Buddy. Kao što smo videli na prethodnim primerima Instrumentation interfejs omogućava dodavanje transformatora čiji je osnovni zadatak transformisanje bajt koda učitanih klijentskih klasa. Ideja transformacije je urađena na niskom nivou, manipulacijom bajt koda što je i osnovni problem koji rešava Byte Buddy biblioteka. Najjednostavnije rečeno ona omogućava generisanje i modifikovanje Java koda u toku izvršavanja aplikacije i time omogućava programerima da jednostavnije, praktičnije i sa manje grešaka manipulišu kodom. Ova biblioteka se koristi u mnogo drugih biblioteka a jedna od najpoznatijih je Mockito u kojem omogućava pravljenje mokovanih objekata i njihovo podešavanje. S obzirom da biblioteka omogućava rukovanje bajt kodom na visokom nivou ona ipak nije onemogućila korisnicima da dodaju ručno delove bajt koda u aplikaciju

## 4. SPECIFIKACIJA SISTEMA

Zadatak obuhvata izradu aplikacije koja je zasnovana na Dropwizard okviru, koji se koristi kao osnova web aplikacije, za REST API i za čuvanje podataka u bazu, dok je front-end realizovan pomoću React okruženja. Korisnik može da se prijavi, odjavi, šalje pozivnice drugim korisnicima, pretraži metrike i entitete i pregleda aktivirane alarme. Ideja projekta je da omogući klijentu da ima bolju preglednost svog sistema.

**Dijagram slučajeva korišćenja**

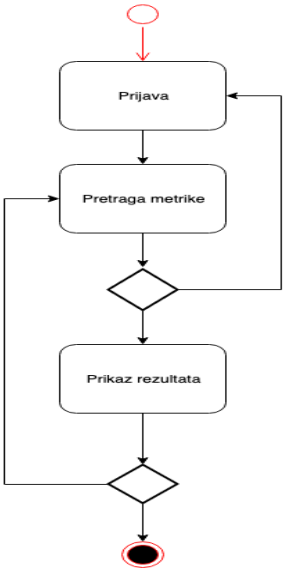
Dijagram slučajeva korišćenja predstavlja sve akcije koje korisnik može da izvrši i prikazan ja na slici 2.




Slika 2. *Dijagram slučajeva korišćenja*

**Dijagram aktivnosti**

Dijagrami aktivnosti su namenjeni modeliranju ponašanja programa. Oni prikazuju sekvencijalne i konkuretne korake u izvršavanju programa. Aktivnosti vezane za korišćenje aplikacije se nalaze na slici 3.




Slika 3. *Dijagram pretrage metrika*

Da bi se mogle koristiti funkcionalnosti aplikacije, korisnik mora biti prijavljen u sistem. Ukoliko pokuša da pristupi stranici za pretragu metrika biće preusmeren na stranicu za prijavu u sistem. Ukoliko je prijava uspešna korisnik se preusmerava nazad na željenu stranicu. Nakon uspešne prijave, korisnik može da pristupi stranicama za pretragu metrika, entiteta ili pregled alarma. Korisnik se može u bilo kom trenutku vratiti na stranicu za prijavu tako što se se odjaviti iz sistema.

## 5. IMPLEMENTACIJA

Projek je u osnovi Java web aplikacija. Dropwizard se koristi kao *back-end,* dakle za implementaciju REST API servis, dok se za *front-end* koristi React klijentska aplikacija. Projekat je razvijen u *Intellij* razvojnom okruženju.

**OSGI bundes**

Instana senzori su u stvari komponente koje su međusobno nezavisne i protokol komunikacije između njih je strogo definisan. Na taj način se sprečava eskaliranje zavisnosti i nameće labava sprega između komponenti. Labava sprega (eng. loose coupling) je pristup povezivanju komponenti u sistemu ili mreži tako da one zavise jedni od drugih u najmanjoj mogućoj meri. Spajanje se odnosi na stepen direktnog znanja koje jedan element ima o drugom.

**Faze senzora**

Svaki senzor se sastoji od dve faze *discovery* i *activation* (eng. aktivacija). Osnovna ideja *discovery* faze je da se provere uslovi koji su potrebni da se bi se aktivirao senzor. Najjednostavniji uslov koji treba da bude zadovoljen da bi se aktivirao Nginx senzor je da postoji proces koji je aktivan sa imenom “nginx: master”.

Nakon detekcije takvog procesa parsira se lista argumenata sa kojom je pokrenut, interpretiraju se ti argumenti kako bi se dobila puna putanja do konfiguracionog fajla u kome se nalaze podešavanja za web server i sve to se prosleđuje Karafu kako bi on aktivirao Nginx senzor i zadao mu podatke koji su mu potrebni. Bitno je napometui da discovery faza se pokreće na svakih 30 sekundi, što omogućava klijentima da ako senzor nije bio pronađen u prethodnom ciklusu bude pronađen u sledećem. Karaf mora biti u mogućnosti da se štiti od pronalazaka koji traju predugo i postoji striktno definisan period vremena od kog *discovery* faza ne sme da bude duža. Karaf ima ograničen broj resursa sa kojima raspolaže i to je glavni razlog za ovako strogu politiku

**Agent**

Osnovna ideja agenta je da služi kao posrednik između senzora i *backenda* i da omogućava zajedničke funkcionalnosti senzorima. Kada se kaže na posredničku konekciju između *backenda* i senzora misli se da svi senzori komuniciraju isključivo sa agentom, potom agent dodaje autorizaciono zaglavlje na poruke i poruke prosleđuje dalje do komponenti *backenda.* Agent podatke sa senzora može da prosleđuje na više kanala. Kanali su: *event* (eng. događaj), *metrics* (eng. metrike) i *tracing* (eng. putanja izvršavanja). Svaki od kanala se drugačije obrađuje a i sama priroda podataka koji se šalju na ova 3 kanala su potpuno drugačijiKonekcija između agenta i backenda je implementirana koristeći HTTP 2 protokol koji između ostaloga omogućava postojanje stalne TCP konekcije kroz koju se šalju paketi. Agent šalje metrike na svaki sekund, kada bi se koritio starti HTTP protokol znatan overhead bi se koristio na uspostavljanje sigurne konekcije svake sekunde. Još jedna prednost novije verzije HTTP protokola je bidirekciona konekcija, u slučaju Instane to se iskorištava na način da backend može da šalje agentima komande koje će izvršiti lokalno i rezultat obrade poslati nazad. Sve to se izvršava preko jedne te iste konekcije

## 6. ZAKLJUČAK

Zadatak ovog rada bio je da se predstavi pregled Instana sistema za nadgledanje aplikacija i infrastrukture, objasni njen značaj i da se pruži detaljniji pregled Javinog Attach API-ja koji omogućava da se Instanin Agent poveže sa korisničkom aplikacijom.

Glavna korist korišćenja Instane jeste ta da korisnik dobija bolji pregled ponašanja celokupnog sistema u realnim uslovima rada aplikacije i na taj način stvara prednost tako što može da vidi koji deo sistema je usko grlo ili ne funkcioniše u skladu sa dizjanom. Na taj način klijent zaokružuje ciklus kontinuiranog dostavljanja koda i njegovog nadzora.

Trenutna mana Instana sistema se krije u deljenom stanju pojedinih backend komponenti koje zbog toga ne mogu dovoljno brzo da skaliraju. Još jedna od mana je i ta da određene tehnologije za koje se rade senzori nemaju backward kompatibilnost što ima reperkusije i na senzore. Posledice su te da je potrebno napisati više senzora za istu tehnologiju u zavisnosti od konkretne verzije instalirane tehonologije.

Dalji razvoj Instana sistema je usmeren na omogućavanje bolje skalabilnosti backend komponenti i većoj stabilnosti senzor komponenti. Pored ova dva pravca značajni su i napori koji se ulažu u sigurnost sistema.

## 7. LITERATURA


[1] Medium article, https://medium.com/refraction-tech-everything/how-netflix-works-the-hugely-simplified-complex-stuff-that-happens-every-time-you-hit-play-3a40c9be254b

[2] APM fundamentls, https://www.apmdigest.com/the-anatomy-of-apm-4-foundational-elements-to-a-successful-strategy

[3] Senonova teorema, https://en.wikipedia.org/wiki/Shannon%E2%80%93Hartley_theorem

[4] Java 2 Platform, Enterprise Edition, http://java.sun.com/j2ee/

[5] T. Schaeck, T. Hepper. Portal Standards. http://www.theserverside.com/articles/article.jsp?l=Portlet_API

[6] Instana dokumentacija, https://docs.instana.io/

[7] Designing Data-Intensive Applications, Martin Kleppman

[8] Building microservices, Sam Newman


**Kratka biografija:**


**Igor Milanović**, rođen 20.04.1992, Rijeka, opština Rijeka, Hrvatska. Završio srednju tehničku školu „Ivan Sarić“ u Subotici. Nakon toga upisao se na Fakultet Tehničkih Nauka, odsek Računarstvo i Automatika.

**Milan Vidaković** je rođen u Novom Sadu 1971. godine. Na Fakultetu tehničkih nauka u Novom Sadu završio je doktorske studije 2003. godine. Na istom fakultetu je 2014. godine izabran za redovnog profesora iz oblasti Primenjene računarske nauke i informatika.