UDK 004.4
Pregledni rad

Adin Jahić, BA
Dr.sc. Nevzudin Buzađija

DEVOPS BUDUĆNOST RAZVOJA SOFTVERA

Sažetak

Softver inženjeru glavna misao predstavlja razvoj sistema koji će omogućiti napredak i olakšati način života ostalih ljudi. Ovaj rad upravo predstavlja meta korak u rješavanju ovog problema na način da će dati inžinjerima, koji imaju zadatak da unaprijede način života ljudi, da kro zmoderne alate razvoja softvera i uvođenje novih koncepata koji su se do prije nekoliko godina vodili kao teoretski i tek u zadnjih par godina implementuju u praksi, zamjene dosadašnji način razvoja softvera u obliku eliminisanja redudantnih procesa u razvoju softvera i dižući nivo eskalabilnosti ,stabilnost i i efikasnosti. Kroz ovaj rad ćemo objasniti apstraktni tip posla koji se naziva „DevOps“,upoznat ćemo se sa osnovnim načinima dizajna,implementacije i isporuke softvera, te objasniti glavne fundamentalne blokove ove oblasti. Koncept cloud computinga, virtualizacije, kontejnerizacija, infrastrukture kao kôda, pravljenja protočnih struktura (eng. pipeline) predstavlja glavni fokus ovog rada, kao i razvoj IDP (interna developer platforma).

Ključne riječi: *Devops, IDP, IT Operacije, CD/CD, Virtualizacija, Kontejnerizacija, Deployment*

UVOD

Za kompletan razvoj softvera se često misli samo da timovi developera vrše razvoj aplikacije bez osvrta na sve veće zahtjeve današnjih sistema. IT operacije predstavljaju ključnu komponentu u rješavanju razvoja projekta. Predstavljaju timove ljudi koji su specijalizovani da posmatraju sistem kao apstraktnu cjelinu i da razmišljaju samo o zahtjevima u cilju razvoja infrastrukture tog sistema. Njihov rad omogućava da isprogramirani sistem razvije svoj puni potencijal. Oni vrše dizajniranje mreže koja je potrebna tom sistemu, vrše implementaciju sigurnosnih mehanizma, prave protokole, uklanjaju redudanciju developerskim timovima. Poboljšanje saradnje između tima operacija i tima developera danas se rješava DevOps pristupom razvoja aplikacija.

1. DEVOPS

DevOps predstavlja skupinu pravila, konvencija, to jeste protokola koji su namijenjeni da povećaju nivo efikasnosti developerskih timova, kao i timova za IT operacije. Uvodeći standarde i konvencije kojih se oba tima moraju držati kako bi efikasnost bila na maksimalnom nivou. Glavni fokus formiranja DevOps grane je formiranje takozvanog IDP (interne developerske platforme) kojom će se služiti oba tima za optimalan razvoj.

DevOps se zasniva na osnovnih 5 postulata[3]:

1. Kolaboracija,
2. Automatizacija,
3. Kontinuiran napredak,
4. Razumijevanje klijenta,
5. Cjelokupnost u implementaciji softvera (gledanje šire slike).

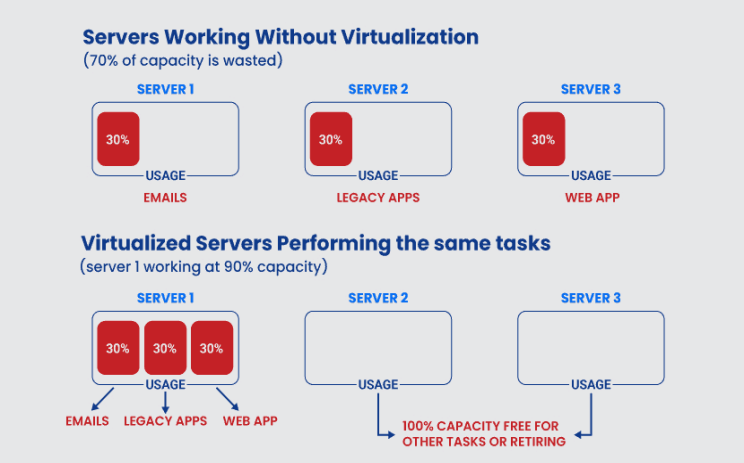
Svi ovi postulati se implementuju kroz razvoj IDP softvera koji omogućava interfejs između developerskih timova i timova IT operacija, omogućava sinhronizaciju tih timova koji uz pomoć projektnih menadžera omogućavaju razvoj softvera na najoptimalnijem nivou i daju najbrže vrijeme odziva klijentu kojem je softver namijenjen.
Trenutno 90% infrastruktura se baziraju na nekoj vrsti clouda. Po nekoj općeprihvaćenoj definiciji Nacionalnog instituta za standardizaciju i tehnologije (NIST) Cloud computing predstavlja: „model za omogućavanje svugdje prisutnog, lako dostupnog mrežnog pristupa jednom zajedničkom skupu računarskih resursa čiji se dijelovi mogu zauzimati i oslobađati sa jako malo manipulacije ili interakcije sa servisnim provajderom“[12].

Karakteristike koje mora da ispunjava svaki Cloud Computing Provider[5]:

- *On-demand self-service* - Svaki provajder mora da omogućiti korisniku da manipulira svojim resursima koje posjeduje bez posredništva tog provajdera. Razlog ovoga je uspostavljanje agilnosti koja je potrebna klijentu da prilagođava svoje resurse komercijalnim zahtjevima koji su mu potrebni.
- *Broad network access* - Servisi i resursi moraju biti dostupni uvijek putem interneta.
- *Resource pooling* - Sve vrste unapređenja i konfiguracije zadanih resursa moraju biti dostupni krajnjem korisniku u bilo koje doba.
- *Rapid elasticity i Measured service* - Mora biti u stanju da u svakom momentu da preciznu metriku krajnjem korisniku da uspješno izvršava sve alokacije resursa koje su mu potrebni.

1.1. Virtualizacija

Virtualizacija je jedna od posljedica koja je nastala razvijanjem Cloud Computinga. U idealnim situacijama omogućava efikasnu upotrebu računarskih resursa. Pošto znamo da svaki računar ima glavne performansne parametre (CPU, Memory, Storage, OS), virtualizacija uzima fizičke parametre i od njih pravi minijaturne „particije“ koje služe kao mali neovisni fizički resursi unutar jednog glavnog. Tom tehnikom ako na primjer naša glavna mašina ima resurse (4CPU, 8GB RAM, 1TB Disk) možemo rastaviti na 4 „virtualno“ neovisne manje mašine koje su resursa (1CPU, 2GB RAM i 256GB Disk), ova 4 računara dijele zajedničke resurse iz mašine koje su nastale ali predstavljaju izolovan sistem na kojem svaki servis može da radi.



Slika 1. *Predstavlja grafičku reprezentaciju ovoga koncepta [13]*

Kao što se može na slici vidjeti postoji veliki broj benefita upotrebe virtualizacije: Manji troškovi, veća efikasnost, brzina konfiguracije, lakši rad.

Svaka virtuelna mašina koja se napravi predstavlja potpuno neovisan sistem koji se može tretirati neovisno o ostalim sistemima. Primjer toga je testiranje sistema gdje naprimjer u razvoju aplikacije se mogu napraviti 3 testna okruženja (VM) gdje će svaki da testira djelovanje aplikacije na različitom operativnom sistemu. Jedna virtuelna mašina može da testira aplikaciju na Windows okruženju dok druga može da radi to isto testiranje na Linuxu, te treća može da simulira Mac OS operativni sistem i gleda ponašanje aplikacije na tom sistemu.

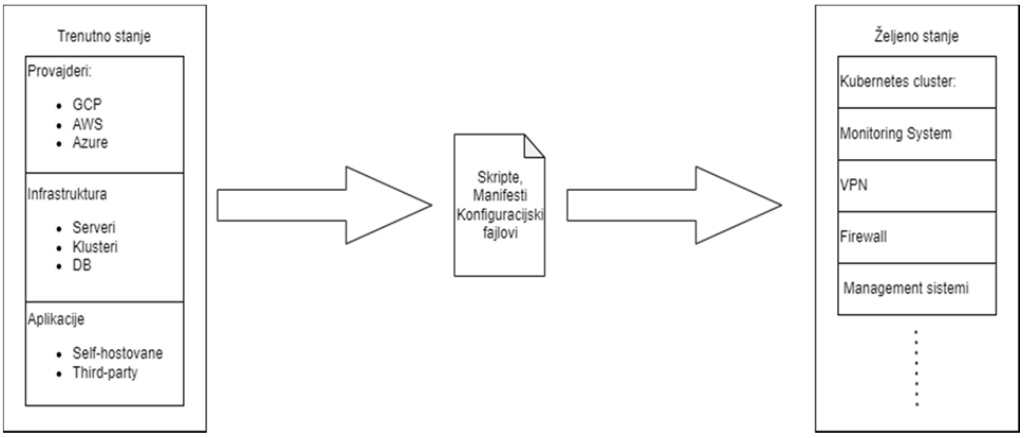
U praksi trenutno definišemo 2 tipa virtuelnih mašina koje možemo da implementujemo na našim sistemima. Dijele se na: hipervizor tipa 1 i hipervizor tipa 2.

Hipervizor predstavlja softverski sloj koji se nalazi između naših virtuelnih mašina i hardvera iz kojih se provizionira. On omogućava da svaka virtuelna mašina dobija dovoljno resursa koji su joj potrebni za dalji rad. Preko njega se vrši sva manipulacija virtuelnim mašinama koja je moguća. Isto tako Hipervizori služe za sinhroniziraju virtuelne mašine na način da ne dozvoljavaju da resursi provizionirani za jednu mašinu budu upotrebljeni od strane druge mašine. Oni vode računa oko sistema umrežavanja, te odlučuju o njihovoj vidljivosti.

„Host OS“, provizionira proizvoljan broj „Guest OS“ kojima korisnik pristupa, a sav taj proces se vrši grafički preko platforme za virtualizaciju.

1.2. Infrastruktura kao kôd

Način na koji moderni inžinjeri DevOps-a implementuju svoja rješenja jeste korištenje potrebnih funkcionalnosti iz veće palete alata koji nam na kraju daju željeni rezultat. Svi manifesti, skripte i konfiguracijski fajlovi se najčešće čuvaju u git repozitorijima, te se kontrolišu putem git sistema na računarima koji se sam po sebi interpretiraju kao program u operativnom sistemu računara. To nam govori da i izvršne skripte, manifesti i ostali proizvodi se interpretiraju i izvršavaju kao niz instrukcija preko komandnog interfejsa kao kôd.



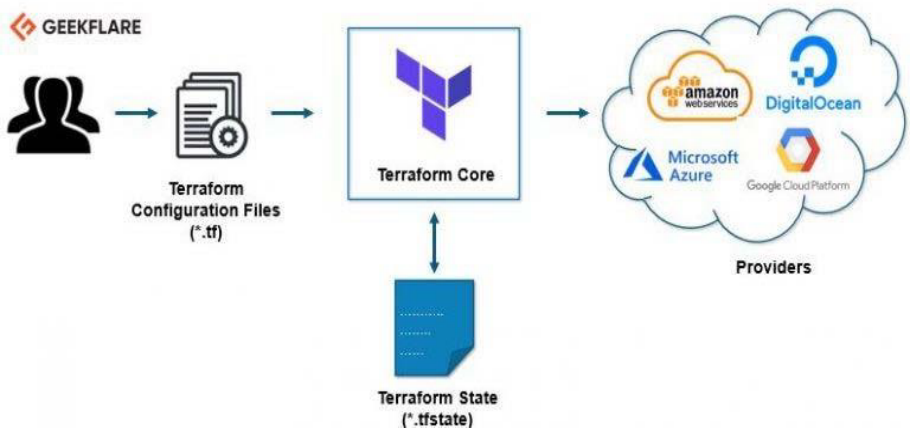
Slika 2. *Šematski prikaz tranzicije stanja [18]*

Razvoj alata za upravljanje konfiguracijama kao što je *Chef, Puppet, Ansible* su bili prvi koji su zauzeli tržište prilikom razvoja. Oni su prvi put pomiješali deklarativne naredbe koje koristimo u programiranju sa problemom koji pokušavamo da riješimo u ovom slučaju, a toje tranzicija između Trenutnog i Željenog stanja.
Oni omogućavaju da korištenjem serijalizacijskih jezika poput YAML pišemo predloške za konfigurisanje, koji od nule već gotov server iskonfigurišu kako bi manuelno svaki inžinjer iskonfigurisao. Primjer ovakvih skripti[10]:

```
hosts: group2 tasks:
name:sshdconfigfilemodifyport lineinfile:

path: /etc/ssh/sshd_config regexp: 'Port 28675'
line: '#Port 22' notify:
restart sshd handlers
name: restart sshd service: sshd
name:sshd
state: restarted
```

Za potpunu upotrebu IaC potreban nam je softver koji je u stanju da rješava probleme uniformnosti i imutabilnosti, da oduzme sav repetitivan i redundantan posao inžinjeru ne treba da svoje vrijeme troši na repetitivne probleme. Trenutno jedini alat na tržištu univerzalno napravljen koji zadovoljava sve ove zahtjeve je **Terraform.**



Slika 3. *Prikaz translacije Trenutnog u Željeno stanje koristećiterraform [7]*

Na ovom dijagramu se jasno vidi proces i uloga koju ima alat poput terraforma u upravljanju konfiguracijom tj. translacijom Trenutnog stanja u Željeno stanje naše infrastrukture.
Da bi bolje izvršili podjelu alata za upotrebu infrastrukture, definisat ćemo četiri glavna zahtjeva koja su potrebna u upravljanju infrastrukturama:

- Inicijalno provizioniranje infrastrukture,
- Upravljanje zadanom infrastrukturom,
- Inicijalnom postavljanje aplikacija,
- Upravljanje postavljenim aplikacijama.

Ovo su osnovi zahtjevi koje alati za upravljanje infrastrukturom trebaju da izvrše. Naime trenutno ni jedan alat na tržištu ne ispunjava sva četiri uslova koja smo naveli.

1.3. CI/CD – Automatska integracija i isporuka

Proces isporuke softvera je u suštini jednostavan. Aplikacija se kompajlira i kompajler izbaci rezultujuće „artifakte“ (JAR, DLL, WAR.). Ti fajlovi se prekopiraju na server i zamijene sa trenutnim verzijama aplikacije koja postoji. Problemi nastaju kada sitne greške koje se dešavaju tokom integracije i isporuke popravljamo na serverima. Rezultat tome je da server s vremenom „mutira“. Server postaje pun svakojakih konfiguracijskih fajlova, skripti, zakrpi. Još veći problem je u tome što u situacijama u kojima imamo više razvojnih okruženja kao npr. (UAT, STG, LIVE) nismo sigurni u funkcionalnost našeg kôda pogotovo jer se ne nalazi u „čistom okruženju“. Dobijemo kao rezultat činjenicu da imamo identičan program na dva servera zbog stvari koje se već nalaze na serveru imamo različito ponašanje aplikacije. Ovaj proces nam daje uvid u činjenicu koliko zapravo je komplikovano bilo vršiti integracije i isporuke kôdova. Jedno od prvih rješenja je formiranje virtuelnih mašina (VM) za svaku izdatu verziju aplikacije. Ovime smo riješili problem nakupljanja svakojakih konfiguracijskih fajlova, zakrpi, skripti koje su bile od prethodnih verzija. Proces se ogledao u tome da se provizionira virtuelna mašina na koju instaliramo potrebne programe i na toj mašini isporučimo aplikaciju. Problem je što prilikom

provizioniranja nakon jako malo vremena dovedemo se u stanje prethodnog haosa jer nakupljanjem zakrpi konfiguracijskih fajlova koji su potrebni da se sistem ponaša očekivano potrebno je s vremena na vrijeme uraditi ispravke i dodatne konfiguracije na serveru. U kratkom vremenskom periodu može doći do situacije gdje se aplikacija na jednom okruženju ne ponaša isto kao na drugom.
Trenutno rješenje se svodi na formiranje sistema za automatizovanu isporuku sofvera. Rezultat je formirana protočna struktura kroz koju naš program treba da prođe dok ne bude aktivan na serveru da ga korisnici mogu da koriste. Da bi razumjeli ovakav kompleksan proces potrebno je napravimo blagu klasifikaciju pojmova za obradu koje ćemo koristiti za dalje objašnjavanje ovog koncepta. Osnovna klasifikacija se svodi na:

- Automatizovanu integraciju,
- Automatizovanu isporuku,
- Automatizovano slanje.

1.4. Automatska integracija (CI)

Jedna od najvećih zabluda prilikom razumijevanja CI/CD protočnih struktura je u nepotpuno razumijevanje ovog prvog koncepta koji se svodi na Automatsku integraciju softvera. Ako ne postoji automatska integracija nije moguće da se izvrši automatska isporuka. Ovo je potpuno logično jer nam je potreban sistem koji će u najmanju ruku da verificira ispravnost kôda koji je potrebno da se isporuči i da označi koju verziju aplikacije je potrebno da isporuči.
Automatska integracija (CI) određuje početno mjesto ali nam ne govori gdje završava. Razlog tome je što u zavisnosti od poslovnih zahjeva i planova različitim firmama potreban različit nivo integracije. Po definiciji ona predstavlja automatsko integrisanje, kompajliranje, i testiranje kôda unutar zadanog okruženja.
Da bi CI potpuno ispravno funkcionisao nije potrebno samo posjedovati repozitorij na koji će developeri gomilati kôd, nego i da će kôd raditi kada dođe period isporuke i slanja kôda na drugo okruženje. Potrebno je kao neki minimum uraditi statičku analizu trenutnog kôda koji se šalje, odraditi (pre-deployment test), provjeriti da li se uopšte može aplikacija kompajlirati, te nakon toga uraditi (post-deployment test) kojim će se verifikovati ispravnost softvera sa ostalim njegovim komponentama.
Automatska integracija, kao što je prije navedeno nema neku završnu tačku kojoj treba da teži. U zavisnosti od firme koja je implementuje, faze integracije će se razlikovati jedna od druge, čisto iz logičkog razloga što nisu svi programi isti i ne trebamo da ih tretiramo kao da su isti. Iz ovog razloga ovaj proces je u praksi jako teško implementirati, ali kada se uspješno implementira daje odlične rezultate koji povećavaju efikasnost na veći nivo[15]
Uopštene faze koje sistem za automatsku integraciju treba da posjeduje:

- Mogućnost slanja kôda na repozitorij,
- Statička analiza,
- Testiranje prije kompajliranja (*Pre-deploymenttest*),
- Kompajliranje kôda,

- Testiranje poslije kompajliranja (*Post-deploymenttest*).

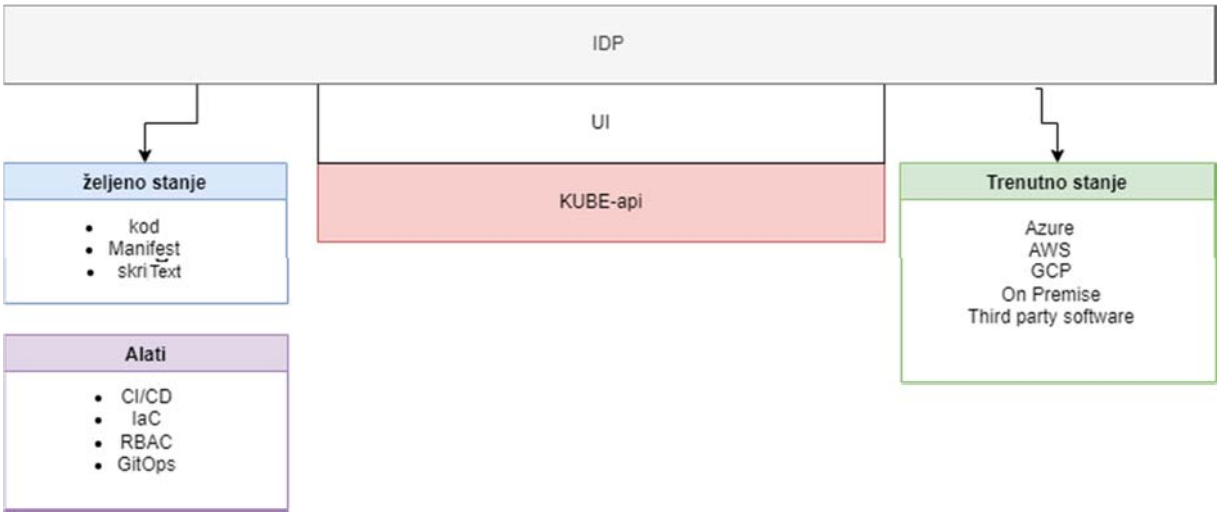
Korisna analogija prilikom izučavanja automatske integracije je da radi na takozvanom *event-based* sistemu. Alati koji služe za automatsku integraciju vrše aktivno slušanje (*event listeners)* koji se pokreću na slanje kôda koje developer izvrši (*trigger)*.

2. INTERNA DEVELOPERSKA PLATFORMA

Servisi koje koriste developeri se integriraju u jedan sistem koji se jednim imenom naziva **IDP (interna developerska platforma)** i oni predstavljaju srž oblasti DevOps-a. Način na koji se dizajniraju IDP-ovi se odlikuje u tome da moramo napraviti servis koji je u stanju da sinhronizuje cijelo Trenutno stanje i Željeno stanje koje imamo u našoj infrastrukuri indiferentno o unutarnjoj strukturu ovih stanja. Potrebno je napraviti spojno mjesto između ova dva stanja.

3. RJEŠENJE I DISKUSIJA

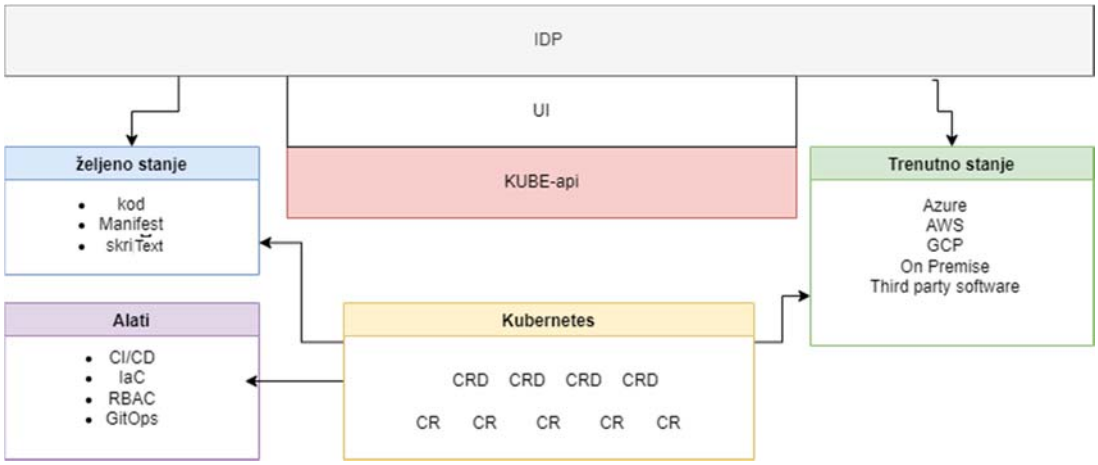
Trenutno na tržištu postoji samo jedan kandidat koji radi na ovom rješenju i trenutno drži monopol u ovom polju a to je Kubernetes KUBEAPI. Ovo je trenutno jedini univerzalni API koji može da integriše sve navedene alate i spoji sve tačke u jedan sistem.



Slika 4. *Grafički prikaz integracije KUBEAPI [18]*

Naime nije dovoljno samo nabacati linkove za različite servise i očekivati od developera da znaju koristiti šta se nalazi na tim linkovima. IDP nije set linkova koji samo predamo developerima kao vreću svega i svačega da oni sebi proberu. Potrebno je eliminisati kompleksnost koju imaju operacije u kojem on samo treba da jednostavno odabere šta želi i da dobije željeni rezultat. U suprotnom bi morao da zna rukovati svakim alatom koji postoji u IDP, a što je nemoguće jer tehnologija previše se brzo razvija.

Glavni zaključak u ovom sistemu je da SVAKO mora biti u stanju da definiše željeno stanje infrastrukture. Svaki developer mora da bude u neovisnom stanju, te da ima dovoljno znanja kako bi mogao provizionirati potrebne resurse koji su mu potrebni za rad, bez zaostajanja od strane tima operacija. Proces upotrebe se ogleda na način da devOps inžinjer napiše predloške za sve moguće tipove resursa koje je moguće provizionirati. To su spomenuti CRD-ovi. Oni predstavljaju definisan izgled resursa koji se mogu provizionirati. Npr: jedna backend instanca se sastoji od baze, aplikacije, api-a, za svaki od ovih resursa je potrebno da se provizionira kontejner koji moraju da budu uvezani u istoj mreži, ispunjeni podacima, sasvim popratnim alatima koji se nalaze na njima.
Developer na osnovu zadanog predloška može iskoristiti CRD da formira svoj CR (custom resource) i može parametrizirati zadani resurs na način da bira jačinu servera,veličinu baze itd. Važno je naglasiti da ima punu kontrolu datom resursu.



Slika 5. *Prikaz šeme čitavog IDP-a [18]*

ZAKLJUČAK

Potrebe tržišta konstantno napreduju, aplikacije postaju kompleksnije, kôdovi postaju teški za praćenje. Timovi IT operacija ne mogu da stižu u obavljanju svih zahtjeva koji stižu. Razvoj IDP je potpuno novi koncept u IT svijetu, a predstavlja obast devOps-a teži. Još uvijek se uzima sa dozom nesigurnosti ali definitivno predstavlja novi način razvoja softvera koristeći moderne tehnike koje nam pruža oblast devOps-a.
DevOps je zasnovan na nizu osnovnih principa kako bi se bolje podudarali sa zahtjevima kupaca u pogledu vremena izlaska na tržište i dodane vrijednosti. DevOps poboljšava komunikaciju i suradnju između poslovnih, razvojnih i operativnih timova. DevOps alati su brojni i uglavnom danas se koriste: Jira, Bitbucket, SonarQube, Artifaktori, IBM Urbancode Deploy itd.
DevOps se razvija u cilju postizanja značajnih IT transformacija koje direktno jačaju poslovne ciljeve. To će pomoći u poboljšanju sposobnosti kompanije da dizajnira, proizvodi, lansira i

održava visokokvalitetna softverska rješenja. Ovi trendovi su katalizatori za stvaranje redovnog i pouzdanog kanala za izdavanje i izgradnju bolje komunikacije između razvojnih, IT i poslovnih timova.

LITERATURA

[1] Darin P. &Viktor F. (2020). The DevOps Toolkit: Kubernetes Chaos Engineering: Kubernetes Chaos Engineering With Chaos Toolkit And Istio, Kindle Edition
[2] Joakim V. (2016). Practical DevOps: Harness the Power of DevOps to Boost Your Skill Set and Make Your IT Organization Perform Better, PACKT PUBLISHING
[3] https://www.atlassian.com/devops/what-is-devops
[4] https://www.baeldung.com/cs/virtualization-vs-containerization
[5] https://binaryterms.com/principles-of-cloud-computing.html
[6] https://devopsdirective.com/
[7] https://www.devopstoolkitseries.com/
[8] https://www.educba.com/types-of-cloud-computing/
[9] https://faun.pub/most-popular-ci-cd-pipelines-and-tools-ccfdce429867
[10] https://www.guru99.com/ansible-tutorial.html
[11] https://medium.com/edureka/cloud-computing-aa3f05f84571
[12] https://www.nist.gov/publications/nist-definition-cloud-computing
[13] https://phoenixnap.com/blog/devops-virtualization
[14] https://www.slideshare.net/mrlemieux66/iaaspaassaas/6
[15] https://technologyconversations.com/
[16] https://xerxes.cs.manchester.ac.uk/comp251/kb/Hypervisor
[17] Mikael K. (2019). Learning DevOps, PACKT PUBLISHING, ISBN 9781838642730
[18] Mitesh S. (2021). Agile, DevOps and Cloud Computing with Microsoft Azure: Hands-On DevOps practices implementation using Azure DevOps, BPB PUBLICATIONS
[19] Viktor F. (2016). The DevOps 2.0 Toolkit: Automating the Continuous Deployment Pipeline with Containerized Microservices, Amazon.com

DEVOPS THE FUTURE OF SOFTWARE DEVELOPMENT

Abstract

As software engineers, our main idea is to develop a system that will enable progress and make the way of life of other people easier. This paper is a meta step in solving this problem by giving engineers the tools to improve people's lives through modern software development tools and introducing new concepts that until a few years ago were considered theoretical and have been implemented only in the last few years. in practice to replace the current way of software development in the form of eliminating redundant processes in software development and raising new scalability, stability, and efficiency. Through this paper we will explain the abstract type of work called "DevOps", we will introduce the basic ways of design, implementation, and deployment of software, and explain the main fundamental blocks of this area. We will go through the concepts of cloud computing, virtualization, containerization, infrastructure as code, and creation of flow structures (pipeline), and the main focus of this paper will be explaining the main point of work of "DevOps" engineers. which is the development of IDP (internal developer platform)

Key words*: DevOps, IDP, ITOperations, CD/CD, Virtualization, Containerization. Deployment*