UDC: 004.738.5:004.2 Info M str. 46-53

RAZVOJ SISTEMA ZA AUTOMATSKO USPOSTAVLJANJE OKRUŽENJA ZA IZVRŠAVANJE WEB APLIKACIJA
BUILDING AN AUTOMATED ENVIRONMENT SETUP TOOL FOR RUNNING WEB APPLICATIONS

Lazarela Antonijević, prof. dr Zoran Đurić

REZIME: U ovom radu analiziran je jedan od ključnih koncepata DevOps-a: automatizacija uspostavljanja okruženja odnosno infrastrukture za izvršavanje web aplikacija u formi konfiguracionih skripti, popularno nazvan "Infrastructure as code". Posebna pažnja posvećena je automatizaciji upravljanja konfiguracijom, pa je dat pregled nekih od najpoznatijih alata za automatsko upravljanje konfiguracijom. Za konkretnu implementaciju razvijen je sistem za automatsko uspostavljanje okruženja za izvršavanje web aplikacija. Kako se sistem bazira na Ansible alatu za upravljanje konfiguracijom, ovaj alat i njegove osobine su detaljnije opisani u radu. Najzad, opisane su glavne odlike konkretnog sistema za automatizaciju i način na koji je realizovan, njegova svrha upotrebe, koncepti koje demonstrira, kao i mogućnosti koje nudi.

KLJUČNE REČI: DevOps, automatizacija, Infrastructure as code, upravljanje konfiguracijom, Ansible

ABSTRACT: This paper analizes one of the key concepts of DevOps: automating the infrastructure needed to run web applications in the form of configuration scripts, commonly known as "Infrastructure as code". The focus is on the configuration management automation and therefore the paper gives a clear review of the most famous configuration management tools. To demonstrate the key concepts, a system for automating the environment setup for running web applications was implemented. Since the system is based on the configuration management tool called Ansible, this tool is described in more detail. Finally, the paper describes the main characteristics of the implemented automation system and how it was developed, its purpose and use case, concepts it demonstrates, as well as the possibilities it offers.

KEY WORDS: DevOps, automation, Infrastructure as code, configuration management, Ansible

1. UVOD

Svaka *web* aplikacija da bi se uspješno izvršavala zahtijeva određene hardverske i softverske resurse. Nakon što se obezbijede adekvatni hardverski resursi i odgovarajući operativni sistem, potrebno je instalirati i konfigurisati softver koji je neophodan za izvršavanje web aplikacije, odnosno konfigurisati okruženje neophodno za izvršavanje *web* aplikacije. Takvo okruženje podrazumijevano se sastoji iz *web* i/ili aplikativnog servera, servera baze podataka, kao i svih biblioteka i sistemskih servisa potrebnih za rad aplikacije. Tako, na primjer, pod pretpostavkom da je obezbijeđen fizički ili virtuelni server i instaliran operativni sistem, prije *deploy*-a jedne jednostavne *web* aplikacije pisane u programskom jeziku PHP koja koristi MySQL bazu podataka, potrebno je instalirati i konfigurisati Apache web/aplikativni server, PHP i MySQL, te, nakon toga, pokrenuti ove servise. Iako se navedeno okruženje može relativno lako manuelno uspostaviti i konfigurisati, proces uspostavljanja okruženja za izvršavanje *web* aplikacija često zna biti dugotrajan, komplikovan, te podložan greškama. Pored toga, ovaj proces se ponavlja iznova i iznova, te postaje rutinski zadatak. Takođe, u tradicionalnom pristupu, puštanje posljednjih izmjena na aplikaciji u produkciju može da potraje sedmicama. Stoga se pojavila potreba za automatizacijom ovog i njemu sličnih procesa.

Automatizacija je već do sada značajno pomogla u rješavanju mnogih inženjerskih problema kao što su prikupljanje podataka, kreiranje izvještaja, ali i uspostavljanje infrastrukture. Sve više razvojnih timova odlučuje se za korištenje alata za automatizaciju upravljanja infrastrukturom. Jedan od glavnih ciljeva automatizacije je oslobađanje IT operativnog tima i razvojnog tima od izvršavanja zadataka koji se ponavljaju, kako bi se mogli fokusirati na druge važne zadatke. Pored toga, očekuje se da sistem za automatizaciju izvodi operacije na pouzdaniji i efikasniji način nego što bi to uradio čovjek i da zahtijeva manje troškova. Prednosti automatizacije ogledaju se u tome što:

- smanjuje kompleksnost nekih procesa i zadataka,
- smanjuje vjerovatnoću pojave greške pri izvršavanju zadataka,
- pomaže da se poboljšaju performanse sistema,
- povećava produktivnost i inovativnost razvojnih i operativnih timova,
- omogućava da se ista aplikacija isporučuje na različita okruženja i
- smanjuje vrijeme isporuke aplikacije.

U ovom radu predstavljen je je namjenski razvijen sistem za automatsko uspostavljanje okruženja za izvršavanje Java EE *web* aplikacije. Ovaj sistem koristi Ansible alat za upravljanje konfiguracijama.

2. INFRASTRUKTURA U OBLIKU KODA

Razvojni tim i IT operativni tim imaju različite zadatke. Razvojni tim je zadužen za razvoj aplikacije, implementaciju novih funkcionalnosti, ispravljanje grešaka u aplikaciji i sl., dok se operativni tim brine za stabilnost aplikacije u produkciji. Ponekad se desi propust u komunikaciji između dva tima, koji može dovesti do toga da okruženje ne odgovara datoj aplikaciji ili da nisu obezbijeđeni potrebni resursi, što dodatno otežava i prolongira puštanje aplikacije u produkciju. U cilju izbjegavanja navednih problema i potrebe za što efikasnijim

isporučivanjem softvera nastaje DevOps (*Development and Operations*), novi pristup koji se razlikuje od tradicionalnog razdvajanja razvoja od isporuke aplikacije, odnosno razvojnog tima od operativnog. Razvoj i isporuka aplikacije bazirani na DevOps principima podrazumijevaju jednostavnije upravljanje čestim izmjenama aplikacije i redovno puštanje aplikacije u produkciju na brži i efikasniji način [1].

Jedan od osnovnih principa DevOps-a je "Infrastruktura u obliku koda", a odnosi se na korištenje konfiguracionih skripti za uspostavljanje željenog okruženja. Nova generacija tehnologija za upravljanje infrastrukturom mijenja dosadašnje načine uspostavljanja i održavanja infrastrukture i njenih dijelova. IaC (*Infrastructure as code*) kao jedan od fundamentalnih koncepata DevOps-a bazira se na principima razvoja softvera. Infrastruktura se tretira kao softver, što omogućava primjenu VCS (*version control*) sistema, automatizovanih biblioteka za testiranje, kao i dobrih praksi u razvoju softvera kao što su TDD (*Test Driven Development*), CI (*continuous integration*) i CD (*continuous delivery*) [2].

IaC alati se najčešće povezuju sa održavanjem servera na *cloud*-u. Međutim, IaC nije isključivo vezan za *cloud*, nego se dati principi i prakse mogu primjeniti i na virtuelne mašine, pa čak i fizički hardver. Primjena IaC principa na fizičke servere, ponekad se žargonski naziva "*bare-metal cloud*". Takođe je moguće koristiti IaC alate na statičkoj infrastrukturi, odnosno infrastrukturi koja je prethodno manuelno uspostavljena i konfigurisana, u svrhu izmjene konfiguracije servera. Mogućnost gašenja i ponovnog podizanja servera je danas veoma bitna, pa se IaC alati najčešće koriste kako za uspostavljanje infrastrukture, tako i za izmjene u konfiguraciji. Mogućnost da se lako uspostavi bilo koji dio infrastrukture otklanja strah od promjena na sistemu, jer se otkazi mogu relativno brzo i lako riješiti. Novi servisi i okruženja se mogu uspostaviti bez imalo napora. Nije potrebno donositi odluke o načinu na koji će se ponovo uspostaviti neki dio infrastrukture, jer se sve potrebne instrukcije nalaze u skriptama koje koriste alati za automatizaciju. Resursi se mogu lako kreirati, uništavati i mijenjati. Sistemi se dizajniraju tako da pretpostave da će se infrastruktura uvijek mijenjati, **što omogućava da** softver nastavi sa izvršavanjem čak i kada se dešavaju promjene na infrastrukturi [2].

Temelj IaC-a su konfiguracione skripte, koje su analogne programskom kodu, pa se mogu pratiti nekim od VCS sistema. Zahvaljujući korištenju konfiguracionih skripti, moguće je obezbijediti konzistentnost među serverima, konstantnu dostupnost servisa, kao i dobro praćenje izmjena i dokumentaciju.

Konfiguracione skripte

IaC se bazira na korištenju konfiguracionih skripti za automatizaciju uspostavljanja i konfigurisanja različitih infrastruktura. Konfiguraciona skipta specifikuje elemente infrastrukture i način na koji infrastruktura treba da se konfiguriše. Alat za automatizaciju koristi instrukcije navedene u fajlu za uspostavljanje i konfiguraciju instanci elemenata infrastrukture. Element infrastrukture može biti server, određeni dio servera, mreža, *load balancer* i drugi. Različiti alati koriste različite nazive za konfiguracione skripte. Tako se konfiguracione skripte koje Ansible koristi nazivaju *playbooks*, Chef koristi "recepte" (eng. *recipes*), a Puppet "manifeste" (eng. *manifests*). Konfiguracione skripte mogu biti napisane u standardnim formatima kao što su JSON (*JavaScript Object Notation*), YAML (*Yet Another Markup Language*) ili XML (*Extensible Markup Language*), a mogu koristiti i sopstveni DSL (*Domain-specific language*). Čuvanje konfiguracije u obliku skripti je mnogo praktičnije od čuvanja u internoj bazi alata, a i omogućava da se skripte tretiraju kao izvorni kod softvera, odnosno da se čuvaju u VCS repozitorijumu i da koriste principe i prakse za razvoj softvera.

Verzionisanje

IaC koristi verzionisanje baš kao i kod razvoja softvera i iskorištava sve mogućnosti koje VCS nudi. VCS pruža pregled istorije promjena na infrastukturi, uvid u to ko je napravio promjene, kada je napravio i zašto (komentari). Ova funkcionalnost je od velike važnosti za detekciju problema i traženje rješenja. U slučaju otkaza, moguće je vratiti infrastrukturu u prethodno stanje. Omogućava lako praćenje skripti, fajlova i datoteka tako što koristi tagove i verzije, što opet pomaže da se otkriju i prevaziđu neki ozbiljniji problemi. VCS može automatski izvršiti određenu akciju kada se desi promjena (eng. *commit*) što je preduslov za CI/CD [2].

Dokumentacija

Pisanje dokumentacije je jedan od izazova za razvojne timove. Dokumentacija treba da bude korisna, relevantna, tačna i ažurna, što zahtijeva određeno vrijeme i napor. Ponekad se dešava da se razvojni tim ne pridržava standarda pisanja dokumentacije, ili standard ne postoji, pa dokumentacija nije razumljiva ili je nedovoljno jasna drugim timovima. Ovo se teško može desiti ako se koriste IaC alati za automatizaciju. IaC zahtijeva minimalnu dokumentaciju, s obzirom na to da su konfiguracione skripte pisane tako da je jasno šta predstavljaju, odnosno one su same sebi dokumentacija [2].

Konzistentnost

IaC omogućava konzistentnost među serverima. Dva aplikativna servera u klasteru koja pružaju iste usluge, treba da budu skoro identični. Njihovi sistemski softveri i konfiguracije treba da budu iste, osim dijelova koji moraju biti različiti kao što su IP adrese. U slučaju da se jedan od elemenata infrastrukture mora izmijeniti, na primjer jedan od servera zahtijeva dodatni prostor na disku, postoje dva načina da se održi konzistentnost. Prvi je da se promijene definicije u konfiguracionim skriptama tako da diskovi imaju više memorije, a drugi je da se taj server izdvoji u posebnu grupu, odnosno da mu se dodijeli drugačija "uloga". Oba načina imaju svoje prednosti i nedostatke, te su daleko su naprednija od izvođenja manuelnih izmjena na serveru [2].

Dostupnost servisa

Mnogim kompanijama je od esencijalne važnosti da servis stalno bude dostupan, nezavisno od toga **šta** se dešava sa infrastrukturom. Ako se jedan server ugasi, drugi server treba da preuzme njegovu funkciju. Ovo je dobro poznat koncept visoke dostupnosti IT servisa, pri čemu automatizacija upravljanja infrastrukturom olak**šava** njegovu realizaciju [2].

3. PREGLED POSTOJEĆIH ALATA

Postoje različite faze uspostavljanja infrastrukture, a na osnovu njih alati za automatizaciju mogu se podijeliti na:

- alate za upravljanje resursima kao što su memorija, procesor, mreža,
- alate za upravljanje konfiguracijom servera (eng. *configuration management*),
- alate za upravljanje infrastrukturom i aplikativnim servisima što uključuje monitoring, upravljanje procesima, *deployment*, itd.

Većina alata nije implementirana isključivo za jednu namjenu, nego obično pokrivaju različite faze uspostavljanja infrastrukture [2]. Neki od najpoznatijih alata koji spadaju u grupu alata za upravljanje konfiguracijom servera su Chef, Puppet, SaltStack i Ansible [5].

Po načinu na koji konfigurišu servere, alati se mogu podijeliti na one koji koriste *pull* model i one koji koriste *push* model.

Kod *pull* baziranih alata, upravljanje konfiguracijom se odvija na sljedeći način:

1. Izmijeni se konfiguraciona skripta na kontrolnom serveru
2. Promjena se šalje nekom centralnom servisu za upravljanje konfiguracijom
3. Agent instaliran na serveru se periodično aktivira
4. Agent se povezuje sa centralnim servisom
5. Agent preuzima novu konfiguracionu skriptu
6. Agent izvršava konfiguracionu skriptu lokalno i mijenja stanje servera

Kod *push* baziranih alata, koraci kod mijenjanja konfiguracije su:

1. Izmijeni se konfiguraciona skripta na kontrolnom serveru
2. Pokreće se konfiguraciona skripta na kontrolnom serveru
3. Kontrolni server se povezuje sa udaljenim serverom
4. Izvršava se konfiguraciona skripta na udaljenom serveru i mijenja se stanje servera

Korake 1 i 2 u oba slučaja izvršava čovjek, dok se ostali koraci izvršavaju automatski. *Push* bazirani model daje bolju kontrolu nad tim kada će promjena da se desi, ali se *pull* model više preferira za skaliranje velikog broja servera i kod uvođenja novih servera koji mogu da se aktiviraju svaki čas [3].

Chef

Chef je alat za upravljanje konfiguracijom i infrastrukturom fizičke ili virtuelne mašine. Razvijen je na bazi Ruby programskog jezika. Može relativno lako da se integriše sa bilo kojom *cloud* tehnologijom. Popularan je kod organizacija čiji je cilj da distribuiraju svoju infrastrukturu na *multi-cloud* okruženje. Chef je integrisan u većinu velikih *cloud* provajdera kao što su Amazon EC2, VMware, OpenStack i drugi [4].

Chef koristi prilično standardno podešavanje kontrolera i udaljenih servera, ali uvodi dodatni koncept "radne stanice" koja predstavlja odvojenu mašinu, najčešće PC. Chef server, radna stanica (eng. *workstation*) i udaljeni serveri (eng. *nodes*) su njegove tri glavne komponente. Chef server čuva podatke za konfiguraciju i upravljanje udaljenim serverima. Chef radna stanica se ponaša kao lokalni repozitorijum, a na njoj se instalira Knife koji se koristi da šalje „knjige recepata" na Chef server. Recepti izvršavaju konkretne zadatke. Udaljeni server komunicira sa Chef serverom i preuzima konfiguracione skripte vezane za taj čvor koje potom izvršava. Ovakav način rada odgovara *pull* modelu upravljanja konfiguracijom. Chef koristi takozvanu *three-tier* klijent arhitekturu.

Chef koristi jedinstvenu terminologiju za opisivanje koncepata automatizacije infrastrukture i upravljanja konfiguracijom koju je preuzeo iz kulinarstva. Koristi termine kao što su "knjiga recepata" (eng. *cookbook*) koja označava kolekciju skripti za automatizaciju ili "kuhanje po receptu" (eng. *following a recipe*) koji označava izvršavanje koda koji kreira neku komponentu infrastrukture. Ovi termini preuzeti su iz kulinarstva u cilju lakšeg razumijevanja i pamćenja, kao i iz marketinških razloga [6].

Puppet

Puppet je alat za automatizaciju razvijen 2005. godine. Sadrži komponente kao što su Collective, Puppet Dashboard, Puppet DB, Hiera i Facter. Koristi se za upravljanje konfiguracijom i orkestraciju. Puppet automatizuje zadatke koji se ponavljaju, omogućava brzo prilagođavanje promjenama i skaliranje servera. Pogodan je za korištenje sa *cloud* okruženjem. Automatizacija kod Puppet-a zasniva se na deklarativnom modelu. Ima četiri glavne faze: definisanje, simulacija, primjena i izvještavanje. Puppet zajednica sadrži mnoštvo različitih gotovih konfiguracionih modula koji se mogu koristiti. Moguće je pomoću jezika za konfiguraciju koji Puppet koristi, napraviti sopstveni modul u skladu sa željenim specifikacijama, koji se kasnije može ponovo iskoristiti.

Podešavanja za Puppet su slična kao i za Chef. Na svakom udaljenom serveru potrebno je instalirati klijenta koji služi da preuzme komande sa kontrolera i izvrši ih.

SaltStack

SaltStack je jedna od najaktivnijih *open source* zajednica koja se veoma brzo razvija. Koristi se od strane velikih IT kompanija i DevOps organizacija u svijetu. Omogućava konfiguraciju i uspostavljanje bilo kojeg tipa infrastrukture na bilo kojoj platformi. Poznat je po paralelnom upravljanju i obezbjeđuje automatizaciju u realnom vremenu..

Salt zahtjeva instalaciju specijalnog servisa kako na kontroleru, tako i na serveru. Svaki udaljeni server preuzima komande sa kontrolera putem komunikacionog *bus*-a i zatim izvršava te komande. Salt takođe ima SSH *push* model sličan Ansible-ovom modelu, pa ima opciju i da se koristi bez instaliranja specijalnog klijenta na udaljenim serverima.

Ansible

Ansible je alat za upravljanje konfiguracijom, ali i za upravljanje *deployment*-om. Podržava i *ad hoc* komande. Ansible koristi SSH i WinRM za komunikaciju sa udaljenim mašinama u zavisnosti od operativnog sistema koji udaljena mašina posjeduje. Ne zahtjeva nikakav dodatni softver za upravljanje. Ne koristi agenta na mašinama pa se kaže da je *agentless* alat. Brz je i jedostavan, lako se instalira i koristi jer ne zahtijeva nikakve dodatne softverske komponente.

Ansible se najjednostavnije podešava jer koristi SSH da se poveže sa udaljenim serverima. Ansible se mora instalirati na kontroleru, dok na serveru nije potrebno instalirati neki specijalni softver. Udaljeni Linux server treba da ima aktivan SSH servis, a Windows server WinRM servis.

Poređenje postojećih alata

Chef i Puppet su pisani u Ruby programskom jeziku, dok su Ansible i SaltStack napisani u Python-u. Ansible i Salt koriste standardni YAML format i Jinja2 *template* jezik, koji su jasni i jednostavni, što čini da se Ansible i Salt lako uče od strane IT stručnjaka bilo kojeg profila. Chef koristi programski jezik Ruby sa proširenim DSL-om. Puppet koristi sopstveni format jezika sličan JSON-u.

Svi osim Ansible-a koriste agente koji se instaliraju na udaljenim mašinama, dok je Ansible jedini *standalone*. Agenti periodično preuzimaju konfiguracione fajlove sa centralnog repozitorijuma i primjenjuju datu konfiguraciju na servere. Dakle, Chef i Puppet koriste *pull* model, dok Ansible i SaltStack koriste *push* model za upravljanje konfiguracijom. Ansible koristi *push* model koji podrazumijeva da centralni server (kontroler) pokreće akcije na serverima koji se konfigurišu. Kontroler se sa udaljenim serverom povezuje putem SSH u slučaju Linux operativnog sistema, a ako se radi o Windows operativnom sistemu koristi se WinRM [11].

Sva četiri alata su pogodna za upravljanje *cloud* okruženjima kao što su EC2, Azure, Digital Ocean i drugi. Takođe, sva četiri alata imaju podršku i za Windows OS.

4. ANSIBLE

Ansible radi sa svakim serverom na koji se moguće povezati, bilo da je udaljen ili lokalni. Ansible kontroler je računar sa kojeg se pokreće skripta i koji se povezuje sa udaljenim serverima. Ansible koristi sopstvenu terminologiju za opisivanje koncepata upravljanja konfiguracijom. Tako se Ansible skripta naziva "playbook" i ona navodi koji "hostovi" (udaljeni serveri) će biti konfigurisani i zadatke (eng. *tasks*) koji će se izvršiti na tim hostovima odnosno način na koji će oni biti konfigurisani. Na primjer, hostovi se mogu označiti sa web1, web2, web3, a taskovi koji će se izvršiti na njima su:

- Instaliranje JDK
- Preuzimanje Wildfly konfiguracije
- Adaptiranje Wildfly konfiguracije
- Instaliranje Wildfly servisa
- Pokretanje Wildfly servisa
- *Deploy* aplikacije

Ansible izvršava ove zadatke na navedenim serverima u isto vrijeme, čeka dok se zadatak završi na svim serverima i potom prelazi na novi zadatak izvršavajući ih redoslijedom kojim su navedeni. Kod upravljanja udaljenim serverima, potrebno je da se poštuje određeni redoslijed kod izvršavanja akcija. Na primjer, potrebno je aktivirati sistem za upravljanje bazama podataka prije aktiviranja web servera. Ansible je napravljen da kontroliše redoslijed akcija. Ansible *playbook* piše se u YAML formatu koji je lako čitljiv i jasan. Instrukcije pisane u YAML su u isto vrijeme i opis (dokumentacija) i kod koji se izvršava.

Ansible radi pomoću modula koji služe da izvode zadatke kao što su instaliranje paketa, restartovanje servisa ili kopiranje konfiguracionog fajla. Ansible moduli su deklarativni, što znači da se koriste da opišu željeno stanje servera. Na primjer, modul *service* može se iskoristiti da navede da *postgresql* servis treba biti pokrenut i omogućen:

```
service: name=postgresql state=started enabled=yes
```

U slučaju da postgresql nije pokrenut *service* modul će ga pokrenuti, u protivnom neće uraditi ništa. Ovaj sistem dovođenja servera u željeno stanje važi za sve module. Ovakvo ponašanje je dobro jer omogućava da se Ansible *playbook* pokrene više puta nad istim serverom i da to ne prouzrokuje greške ili neke neželjene efekte.

Ansible *playbooks* nisu namijenjeni da budu korišteni u različitim kontekstima. Međutim, postoje načini kako se Ansible skripte mogu napraviti takvim da se mogu ponovo koristiti u različitim kontekstima, a jedan od tih načina je uvođenje Ansible "uloga" (eng. *roles*). Ansible uloge služe da podijele *playbook* u više fajlova i time pojednostavljuju pisanje kompleksnih skripti, ali i omogućavaju ponovnu upotrebu. Ansible uloga se može posmatrati kao nešto što se dodijeli serveru u zavisnosti od njegove namjene. Na primjer, serveru koji će služiti kao server baze podataka, dodijeliće se uloga servera baze podataka koja će sadržati sve potrebne skripte, fajlove, varijable i meta podatke u vezi sa datom ulogom (bazom podataka) [3]. Ansible Galaxy je online repozitorijum koji sadrži Ansible uloge koje se mogu preuzeti i koristiti.

Kod pojedinih alata za upravljanje konfiguracijom, moguće je istu skriptu koristiti za upravljanje serverima koji imaju različite operativne sisteme. To znači da je format instrukcija u skripti platformski nezavisan i podignut na viši nivo apstrakcije. Na primjer, umjesto navođenja konkretnog paket menadžera kao što je "apt" ili "yum", koristi se apstrakcija "package". Ansible ne funkcioniše na ovaj način, nego je zavisan od operativnog sistema udaljenog servera. Dakle, poznavanje osnova pisanja *shell* skripti neophodno je za korištenje Ansible alata. Međutim, očigledna prednost Ansible-a i odnosu na *shell*

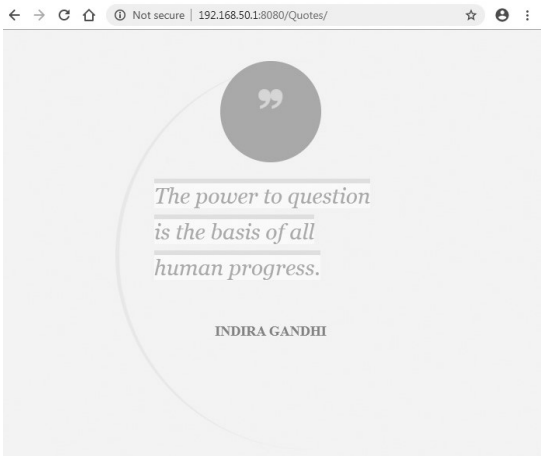
skripte je YAML jezik koji je lakše čitljiv i razumljiviji od klasičnih *shell* komandi. Takođe, Ansible i drugi alati za upravljanje konfiguracijom imaju jednu važnu osobinu koja predstavlja veliku prednost u odnosu na *shell* skripte, a to je idempotencija (eng. *idempotence*). Ova osobina podrazumijeva da se Ansible skripta može izvršiti više puta bez rizika od nepredviđenih rezultata i grešaka. Idempotentna komanda će provjeriti da li je sistem u željenom stanju i samo ako nije izvršiti akciju da dovede sistem u dato stanje. To znači da se ista skripta može sigurno koristiti i kasnije nakon izvršenih izmijena u konfiguraciji servera, a ne samo za inicijalnu konfiguraciju servera kao što je to najčešće slučaj sa klasičnim *shell* skriptama. Korištenjem klasičnih *shell* skripti automatizuje se inicijalna faza konfiguracije servera, ali se kasnije najčešće izmjene u konfiguraciji i održavanje servera vrše manuelno [5].

5. AUTOMATSKO USPOSTAVLJANJE OKRUŽENJA ZA IZVRŠAVANJE JAVA EE WEB APLIKACIJE

Kako bi se Java EE web aplikacija mogla izvršavati u produkcionom okruženju, nepohodno je isto i uspostaviti. Ovakvo okruženje se, minimalno, sastoji se iz sljedećih komponenti: *web* aplikacije, baze podataka koja se nalazi na odgovarajućem sistemu za upravljanje bazama podataka, aplikativnog servera, konfiguracionih fajlova za uspostavljanje okruženja za izvršavanje date *web* aplikacije i njen *deployment*. U ovoj sekciji demonstriran je proces uspostavljanja okruženja za izvršavanje Java EE web aplikacije uz pomoć Vagrant mašine koja predstavlja kontroler i služi za izvršavanje datih konfiguracionih fajlova.

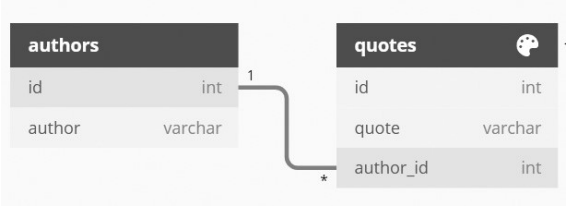
Vagrant je softver za automatsko kreiranje i upravljanje virtuelnim mašinama. Konfiguracija virtuelne mašine navodi se u skripti koja se naziva *Vagrantfile*. Na osnovu specifikacije navedene u Vagrant fajlu, automatski se kreira virtuelna mašina sa odgovarajućim operativnim sistemom i instalira se odgovarajući softver, što je od velike koristi ako je potrebno na brz i jednostavan način uspostaviti virtuelno okruženje.

Web aplikacija *QuotesApp* je jednostavna *web* aplikacija pisana u programskom jeziku Java koja pri svakom pokretanju prikazuje citat, slučajno izabran iz baze podataka (Slika 1). Kod aplikacije se pomoću Maven alata pakuje u *artifact* u *war* formatu koji se objavljuje na WildFly aplikativnom serveru.



Slika 1: Web aplikacija *QuotesApp*

Aplikacija koristi PostgreSQL sistem za upravljanje bazama podataka. Baza se sastoji iz dvije tabele koje sadrže citate i njihove autore povezane po identifikatoru autora. Šema baze podataka prikazana je na slici 2.



Slika 2: Šema baze podataka aplikacije *QuotesApp*

Okruženje za izvršavanje *web* aplikacije uspostavlja se automatski pomoću skripti pisanih u Ansible alatu za upravljanje konfiguracijom. Ansible *playbook* se sastoji iz dvije „predstave“ (eng. *plays*) od kojih jedna služi za konfigurisanje servera baze podataka, a druga za konfiguraciju aplikativnog servera.

Kao Ansible kontroler koristi se virtuelna mašina opisana Vagrant fajlom (Slika 3). Vagrant ima ugrađenu podršku za Ansible u obliku Ansible *provisioner*-a i Ansible *local provisioner*-a [7], čija upotreba je demonstrirana u navedenom fajlu na slici 3.

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"

  config.vm.network "private_network", ip: "192.168.50.4"

  config.vm.provision "shell", inline: <<-SHELL
    yum install epel-release -y
    yum -y update
    yum -y install python-pip pip
    pip install ansible==2.5.1
    pip install pywinrm
  SHELL

  config.vm.provision "ansible_local" do |ansible|
    ansible.playbook = "playbook.yml"
    ansible.limit = "all"
    ansible.inventory_path = "inventory"
    ansible.install = false
  end
end
```

Slika 3: *Vagrantfile*

Shell sekcija fajla služi za instalaciju Ansible alata i potrebnih paketa, kao npr. *pywinrm* paketa koji je neophodan da bi kontroler uspostavio *winrm* konekciju sa Windows hostom. Zatim se specifikuje putanja i naziv *playbook*-a (u ovom slučaju to je *playbook.yml*), *inventory* fajl i limit koji se odnosi na hostove navedene u tom fajlu, kao i opcija *ansible.install* koja u ovom slučaju ima vrijednost *false*, jer nema potrebe da se Ansible automatski instalira, s obzirom na to da je njegova instalacija navedena ranije, u istom fajlu. U slučaju da je vrijednost ove varijable *true*, Vagrant bi automatski instalirao najnoviju verziju Ansible-a.

Baza podataka se konfiguriše na serveru označenom sa *db*, a aplikativni server na hostu *win*. U praktičnom primjeru za server na kojem se izvršava aplikacija korištena je Windows mašina, a za server baze podataka Centos/Linux radi demonstracije rada sistema na različitim platformama. U skladu s tim korišteni su adekvatni Ansible moduli, tako na primjer svi Ansible moduli za Windows imaju prefiks *win*. *Inventory* fajl (Slika 5) sadrži opise servera, njihovu klasifikaciju, IP adrese, tipove konekcije i ostalo [8]. *Inventory* fajl treba da se nalazi u direktorijumu zajedno sa *Vagrantfile*-om i *playbook.yml* i da se navede u *Va-*

grantfile varijabli ansible.*inventory_path*. Takođe je obavezno navesti *limit* koji predstavlja servere koji će biti konfigurisani. Može imati vrijednost *all* za sve servere navedene u *inventory* ili naziv specifične grupe kojoj serveri pripadaju.

```
[win]
172.25.180.140

[win:vars]
ansible_user=antonijevicl
ansible_password=4FcG1hpmS64
ansible_connection=winrm
ansible_winrm_server_cert_validation=ignore
ansible_winrm_transport=ntlm
ansible_winrm_message_encryption=auto

[db]
127.0.0.1

[db:vars]
ansible_connection=local
```

Slika 4: *Inventory*

Konfiguracija sistema za upravljanje bazama podataka

Prije izmjena konfiguracije sistema za upravljanje bazama podataka instalira se *postgresql* paket i kreira *cluster* sistema za upravljanje bazama podataka (Slika 6). Zatim se mijenja konfiguracija tako da se omogući *remote* konekcije na bazu. Podrazumijevano ponašanje *postgresql* servera je da dozvoljava isključivo lokalne konekcije. Ono se može promijeniti tako što se izmijene *postgresql* konfiguracioni fajlovi. Na slici 7 vidi se da se za lokalne konekcije dozvoljava pristup svim bazama za korisnika „postgres“ bez autentifikacije putem lozinke (*trust*), dok se dozvoljavaju *remote* konekcije sa bilo koje mreže, ali isključivo na *quotes* bazu podataka i to samo za korisnika „postgres“, gdje se koristi autentifikacija pomoću lozinke koja se preko mreže šalje heširana md5 algoritmom [9]. U nekom realnom produkcionom okruženju trebalo bi uvesti dodatne restrikcije u vezi sa udaljenim pristupom bazi podataka, kao što su ograničenje pristupa isključivo sa adrese na kojoj se nalazi aplikativni server. U slučaju da je omogućen *firewall*, da bi se pristupilo bazi podataka, potrebno je otvoriti port na kojem radi PostgreSQL servis. Na slici 7 vidi se da je port otvoren isključivo za pristup sa adrese na kojoj se nalazi aplikativni server, što povećava sigurnost sistema.

```
---
- name: Database
  hosts: db
  vars:
    db_name: quotes
  vars_files:
    - secrets.yml
  tasks:
  - name: Install packages
    yum:
      name:
        - python-psycopg2
        - postgresql-server
        - postgresql-contrib
      state: present
    become: yes

  - name: Check if data cluster is created
    stat:
      path: /var/lib/pgsql/data/pg_hba.conf
    register: pgdata
    become: yes

  - name: Create the data cluster
    shell: postgresql-setup initdb
    become: yes
    when: pgdata.stat.exists == false
```

Slika 5: *Instalacija PostgreSQL*

Nakon što je konfiguracija izmijenjena, pokreće se *PostgreSQL* servis i kreira se baza podataka „quotes“, nad kojom se zatim izvršava *sql* skripta koja se kopira sa kontrolera na server i sadrži *sql* kod za dodavanje tabela u bazu i dodavanje podataka (Slika 8). *Sql* skripta prikazana je na slici 9. Ovim je konfiguracija sistema za upravljanje bazama podataka završena.

```
- name: Ensure the PostgreSQL service is running
  service: name=postgresql state=started enabled=yes
  become: yes

- name: Change default password for user postgres
  shell: "psql -U postgres -c \"ALTER USER postgres PASSWORD '{{ db_password }}';\""

- name: Ensure database is created
  postgresql_db: name={{ db_name }}
                 encoding='UTF-8'
                 lc_collate='en_US.UTF-8'
                 lc_ctype='en_US.UTF-8'
                 template='template0'
                 state=present

- name: Copy db.sql from controller to node
  copy:
    src: db.sql
    dest: quotes.sql

- name: Insert quotes to database
  shell: 'psql -U postgres -f quotes.sql {{ db_name }}'
```

Slika 7: *Kreiranje i manipulacija „quotes“ bazom*

```
- name: Modify pg_hba.conf to allow postgres user on the local system to connect to any database using Unix-domain sockets
  lineinfile:
    path: /var/lib/pgsql/data/pg_hba.conf
    regexp: '^local*'
    line: 'local   all            postgres                                trust'
  become: yes

- name: Modify pg_hba.conf to allow connections from anywhere else on the Internet
  lineinfile:
    path: /var/lib/pgsql/data/pg_hba.conf
    line: 'host    quotes         postgres            0.0.0.0/0          md5'
  become: yes

- name: Modify postgresql.conf to allow connections from anywhere else on the Internet
  lineinfile:
    path: /var/lib/pgsql/data/postgresql.conf
    regexp: '^#?listen_addresses*'
    line: "listen_addresses = '*'"
  become: yes

- name: Open port 5432 to specific IP address
  firewalld:
    zone: public
    rich_rule: 'rule family="ipv4" source address="{{ application_server_ip_address }}" port protocol="tcp" port="5432" log prefix="test-firewalld-log" level="info" accept'
    permanent: true
    immediate: true
    state: enabled
  become: yes
```

Slika 6: *Konfiguracija PostgreSQL*

```
DROP TABLE IF EXISTS quotes;
DROP TABLE IF EXISTS authors;

CREATE TABLE authors (id serial PRIMARY KEY, author varchar);
CREATE TABLE quotes
(
    id serial PRIMARY KEY,
    quote varchar,
    author_id integer,
    CONSTRAINT author_id_fkey FOREIGN KEY (author_id)
      REFERENCES authors (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
);

INSERT INTO authors (id, author) VALUES (1, 'John Keats');
INSERT INTO authors (id, author) VALUES (2, 'Ernest Hemingway');
INSERT INTO authors (id, author) VALUES (3, 'Franklin D. Roosevelt');
INSERT INTO authors (id, author) VALUES (4, 'Heraclitus');
INSERT INTO authors (id, author) VALUES (5, 'Indira Gandhi');
INSERT INTO quotes (quote, author_id) VALUES ('I love you the more in that I believe you had liked me for my own sake and for nothing else.', 1);
INSERT INTO quotes (quote, author_id) VALUES ('But man is not made for defeat. A man can be destroyed but not defeated.', 2);
INSERT INTO quotes (quote, author_id) VALUES ('When you reach the end of your rope, tie a knot in it and hang on', 3);
INSERT INTO quotes (quote, author_id) VALUES ('There is nothing permanent except change.', 4);
INSERT INTO quotes (quote, author_id) VALUES ('You cannot shake hands with a clenched fist.', 5);
INSERT INTO quotes (quote, author_id) VALUES ('There is not love where there is no will.', 5);
INSERT INTO quotes (quote, author_id) VALUES ('The power to question is the basis of all human progress.', 5);
INSERT INTO quotes (quote, author_id) VALUES ('The eyes are more exact witnesses than the ears.', 4);
INSERT INTO quotes (quote, author_id) VALUES ('You cannot step into the same river twice.', 4);
INSERT INTO quotes (quote, author_id) VALUES ('There is no friend as loyal as a book.', 2);
```

Slika 8: *Skripta za manipulaciju bazom*

Konfiguracija aplikativnog servera

Play za konfiguraciju servera za izvršavanje *web* aplikacije sadrži *task* koji instalira JDK u slučaju da nije već instaliran, a zatim preuzima WildFly konfiguraciju. *Task* za instalaciju i konfiguraciju JDK je odvojen u zaseban *yml* fajl radi bolje preglednosti. Dodatna konfiguracija koja služi da omogući instaliranje Wildfly-a kao Windows servisa kopira se na server [10]. Izmijenom *standalone.xml* fajla u instalaciji Wildfly servera, vrše se potrebne izmjene u konfiguraciji servera, kao što su promjena podrazumijevanog ponašanja da se servisu pristupa isključivo sa *localhost* adrese. Vrši se *deploy* aplikacije tako što se *war artifact* kopira u *deployments* folder na serveru i pokreće se server kao Windows servis. Najzad se otvaraju odgovarajući portovi na kojima radi aplikacija (Slika 9).

```
- name: Firewall rule to allow connections on TCP port 8080
  win_firewall_rule:
    name: Allow WildFly port 8080
    localport: 8080
    action: allow
    direction: in
    protocol: tcp
    state: present
    enabled: yes

- name: Firewall rule to allow connections on TCP port 8443
  win_firewall_rule:
    name: Allow WildFly port 8443
    localport: 8443
    action: allow
    direction: in
    protocol: tcp
    state: present
    enabled: yes
```

Slika 9: *Otvaranje portova 8080 i 8443*

Jednostavnom komandom *vagrant up* pokreće se Ansible kontroler koji izvršava dati *playbook* i ispisuje poruke na konzolu. *Playbook* se može ponovo pokrenuti komandom *vagrant provision --provision-with ansible_local*. Ansible radi tako da u slučaju da je neki zadatak (*task*) već izvršen, on ga preskače i ispisuje poruku *ok*, a ako nije prethodno izvršen ispisuje poruku *changed*, dok poruka *skipping* znači da se izvršenje zadatka preskače jer je tako navedeno uslovom u skripti (Slika 10).

```
PLAY [Deploy QuotesApp] ********************************************************

TASK [Gathering Facts] *********************************************************
ok: [192.168.50.1]

TASK [Check if Java is installed] **********************************************
ok: [192.168.50.1]

TASK [Download JDK] ************************************************************
skipping: [192.168.50.1]

TASK [Install Java] ************************************************************
skipping: [192.168.50.1]

TASK [Set Java_home] ***********************************************************
skipping: [192.168.50.1]

TASK [Add Java to path] ********************************************************
skipping: [192.168.50.1]

TASK [Check if Wildfly is downloaded] ******************************************
ok: [192.168.50.1]

TASK [Download Wildfly zip] ****************************************************
skipping: [192.168.50.1]

TASK [Unzip Wildfly] ***********************************************************
skipping: [192.168.50.1]

TASK [Deploy QuotesApp.war] ****************************************************
ok: [192.168.50.1]

TASK [Copy service configuration] **********************************************
ok: [192.168.50.1]

TASK [Bind Wildfly to specific IP address] *************************************
ok: [192.168.50.1]

TASK [Install Wildfly as service] **********************************************
changed: [192.168.50.1]

TASK [Start Wildfly as service] ************************************************
changed: [192.168.50.1]

TASK [Firewall rule to allow connections on TCP port 8080] *********************
ok: [192.168.50.1]

TASK [Firewall rule to allow connections on TCP port 8443] *********************
ok: [192.168.50.1]

PLAY RECAP *********************************************************************
127.0.0.1                  : ok=12   changed=3    unreachable=0    failed=0
192.168.50.1               : ok=10   changed=2    unreachable=0    failed=0
```

Slika 10: *Prikaz izvršenja play-a*

Prednosti razvijenog sistema

Prednosti korištenja automatskog sistema za uspostavljanje okruženja za izvršavanje *web* aplikacije u odnosu na manuelno uspostavljanje istog okruženja su očigledne. Potrebne akcije se izvršavaju automatski bez ljudske interakcije, što otklanja mogućnost grešaka izazvanih ljudskim faktorom, a koje se tiču unosa pogrešnih komandi, izvršavanja akcija u pogrešnom redoslijedu i ostalih nepredviđenih akcija izazvanih nepažnjom. Jedna od važnijih prednosti je što su sve izvršene akcije navedene u skriptama, što ujedno predstavlja i kod za izvršavanje i dokumentaciju. Vrijeme koje bi se inače utrošilo na pisanje dokumentacije, sada je zamijenjeno vremenom potrebnim za razvoj sistema. Sistem je baziran na Ansible alatu za upravljanje konfiguracijom, pa ako se dobro savlada Ansible alat, sistem se može lako prilagoditi i za uspostavljanje nekog drugog okruženja za izvršavanje neke druge *web* aplikacije.

6. ZAKLJUČAK

Da bi se primjenio DevOps pristup razvoju i isporuci aplikacija, neophodno je automatizovati uspostavljanje infrastrukture i konfiguraciju okruženja u formi koda odnosno skripti sa kodom. Date skripte mogu da služe da instaliraju operativni sistem, instaliraju i konfiguriše različite instance servera, instaliraju i konfigurišu poteban softver i potrebne servise. Koristeći skripte za automatizaciju, ista konfiguracija može se primjeniti na hiljade servera u isto vrijeme.

IaC kao jedan od osnovnih principa DevOps-a odnosi se na korištenje konfiguracionih skripti za uspostavljanje željenog okruženja za izvršavanje aplikacija. IaC podrazumijeva automatizaciju uspostavljanja infrastrukture na način da se instrukcije za uspostavljanje i upravljanje tretiraju kao programski kod koji se kompajlira i interpretira. Takav kod može da sadrži definicije servera, mreže i ostalih elemenata infrastrukture, instrukcije za instaliranje paketa, pokretanje servisa i za *deployment* aplikacije. Uspostavljanje infrastrukture može da se odnosi na upravljanje resursima kao što su memorija, procesor, mreža, konfigurisanje fizičkih uređaja, instaliranje softverskih paketa, upravljanje konfiguracijom servera, pokretanje servisa, monitoring, *deployment*, itd.

Fokus ovog rada je na fazi upravljanja konfiguracijom servera. Kada se spominje upravljanje konfiguracijom obično se misli na opisivanje željenog stanja servera u formi instrukcija i korištenja nekog alata za primjenjivanje datih instrukcija i dovođenje servera u željeno stanje. To stanje podrazumijeva da su potrebni paketi instalirani, odgovarajući servisi pokrenuti, konfiguracioni fajlovi sadrže prave vrijednosti i permisije, itd. Neki od najpoznatijih alata za upravljanje konfiguracijom su Chef, Puppet, SaltStack i Ansible.

Baziran na Ansible alatu za upravljanje konfiguracijom, za potrebe ovog rada razvijen je sistem za automatsko uspostavljanje okruženja za izvršavanje web aplikacije pisane u programskom jeziku Java koja koristi PostgreSQL sistem za upravljanje bazama podataka. Dati sistem može se koristiti u bilo kojoj organizaciji, kako za testna, tako i za produkciona okruženja. Sistem demonstrira automatizaciju najčešćih zadataka potrebnih za uspostavljanje okruženja za izvršavanje *web* aplikacije na različitim platformama. Prednosti korištenja razvijenog sistema u odnosu na manueno uspostavljanje okruženja su automatizacija zadataka koji se ponavljaju, smanjeno vrijeme isporuke aplikacije, smanjena mogućnost grešaka, korištenje konfiguracionih skripti, lako isporučivanje aplikacije na više servera i mnoge druge.

LITERATURA

[1] Michael Hüttermann. *DevOps for Developers*, 2012
[2] Kief Morris. *Infrastructure as Code*, pages 3 – 21, O'Reilly Media, Inc., 2016.
[3] Lorin Hochstein. *Ansible: Up and Running*, O'Reilly Media, Inc., 2015
[4] Rishabsh Sharma, Mitesh Soni. *Learning Chef*, Packt Publishing, 2015
[5] Matt Jaynes, *Taste test: Puppet, Chef, SaltStack, Ansible*, 2015
[6] Mischa Taylor and Seth Vargo. *Learning Chef*, O'Reily Media, Inc., 2015.
[7] https://www.vagrantup.com/docs/provisioning/ansible_local.html, posjećivano jula 2019
[8] https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html, posjećivano jula 2019
[9] https://www.postgresql.org/docs/9.1/auth-pg-hba-conf.html
[10] https://github.com/wildfly/wildfly-core/tree/master/core-feature-pack/src/main/resources/content/docs/contrib/scripts/service, posjećivano jula 2019
[11] https://www.infoworld.com/article/2609482/data-center-review-puppet-vs-chef-vs-ansible-vs-salt.html, posjećivano jula 2019

Lazarela Antonijević, dipl. ing, Elektrotehnički fakultet, Univerzitet u Banjoj Luci, RS, BiH
Kontakt: lazarela.antonijevic@gmail.com
Oblast interesovanja: objektno-orijentisano programiranje i modelovanje, Internet programiranje, informacioni sistemi, DevOps, sigurnost na Internetu

prof. dr Zoran Đurić, Elektrotehnički fakultet, Univerzitet u Banjoj Luci, RS, BiH
Kontakt: szoran.djuric@etf.unibl.org
Oblast interesovanja: sigurnost, kriptografija, PKI, platni sistemi i protokoli, formalna verifikacija, mašinsko učenje, data science, objektno-orijentisano programiranje i modelovanje, Internet programiranje, razvoj mobilnih aplikacija, XML-bazirana međuoperativnost, Web servisi, računarske mreže, penetration testing, sistem integracija